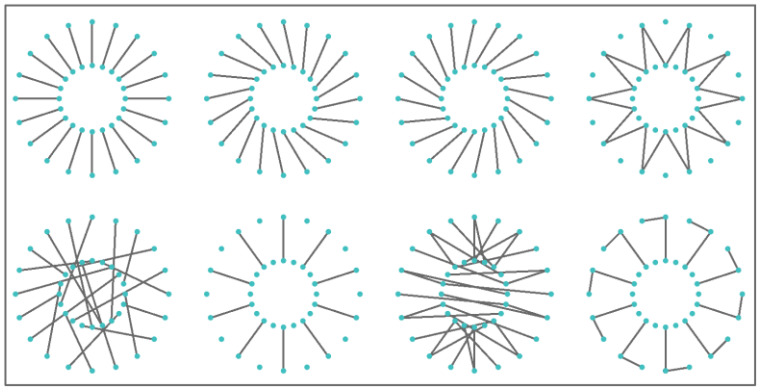


Algoritmos Esenciales y Estructuras de Datos

para Diseño Computacional en Grasshopper

Primera edición

Rajaa Issa

Robert McNeel & Associates

Algoritmos Esenciales y Estructuras de Datos para Diseño Computacional, Primera Edición, de Robert McNeel & Associates, 2020, tiene una licencia de Creative Commons Attribution-Share Alike 3.0 en Estados Unidos.

Traducción al español: Noemí Fluixà (noemi@mcneel.com)

Contenido

Introducción 4

Capítulo 1: Algoritmos y datos 5

1_1: Diseño algorítmico 5

1_2: Partes de algoritmos 6

1_3: Diseño de algoritmos: el proceso de 4 pasos 7

1_4: Datos 12

1_5: Fuentes de datos 12

1_6: Tipos de datos 13

1_7: Procesamiento de datos 16

1_7_1: Operaciones numéricas 16

1_7_2: Operaciones lógicas 18

1_7_3: Análisis de datos 19

1_7_4: Ordenación 19

1_7_5: Selección 19

1_7_6: Mapeado 20

1_8: Dificultades del diseño algorítmico 24

1_8_1: Tipo de entrada no válido o incorrecto 24

1_8_2: Entrada no intencionada 25

1_8_3: Orden de funcionamiento incorrecto 26

1_8_4: Estructuras de datos no coincidentes 26

1_8_5: Tiempo de procesamiento largo 27

1_8_6: Mala organización 28

1_9: Tutoriales de algoritmos 28

1_9_1: Tutorial de círculos unidos 28

1_9_2: Tutorial de esfera con límites 30

1_9_3: Tutorial de operaciones de datos 31

1_9_4: Tutorial para resolver dificultades 32

Capítulo 2: Introducción a las estructuras de datos 34

2_1: Resumen 34

2_2: Generación de listas 35

2_3: Operaciones con listas 38

2_4: Correspondencia de listas 42

2_5: Tutoriales de estructuras de datos 46

2_5_1: Tutorial de un tubo de grosor variable 46
2_5_2: Tutorial de correspondencia personalizada 48
2_5_3: Tutorial de una celosía sencilla 49
2_5_4: Tutorial de un collar de perlas 52
Capítulo 3: Estructuras de datos avanzadas 55
3_1: La estructura de datos de Grasshopper 55
3_1_1 Introducción 55
3_1_2 Procesamiento de árboles de datos 55
3_1_3 Notación de árboles de datos 57
3_2: Generación de árboles 59
3_3: Correspondencia de árboles 62
3_4: Cruzar árboles 65
3_5: Operaciones básicas con árboles 67
3_5_1: Visualizar la estructura del árbol 67
3_5_2: Operaciones de lista con árboles 67
3_5_3: Insertar de listas en árboles 69
3_5_4: Aplanar de árboles a listas 70
3_5_5: Combinación de flujos de datos 71
3_5_6: Invertir la estructura de datos 71
3_5_7: Simplificar la estructura de datos 73
3_6: Operaciones avanzadas con árboles 76
3_6_1: Elementos relativos 76
3_6_2: Dividir árboles 81
3_6_3: Mapeador de ruta 86
3_7: Tutoriales de estructuras de datos avanzadas 92
3_7_1: Tutorial de cubierta inclinada 92
3_7_2: Tutorial de triángulos en diagonal 95
3_7_3: Tutorial de zigzag 96
3_7_4: Tutorial de tejido 97

Introducción

Estructuras de Datos y Algoritmos Esenciales para Diseño Computacional presenta metodologías efectivas para desarrollar algoritmos complejos de modelado 3D utilizando Grasshopper. También aborda ampliamente la estructura de datos adoptada por Grasshopper y sus principales herramientas de organización y gestión.

El material está enfocado a diseñadores interesados en el diseño paramétrico y con poca o ninguna experiencia en programación. Todos los conceptos son explicados visualmente usando Grasshopper® (GH), el entorno de modelado generativo para Rhinoceros® (Rhino). Este libro es una guía para principiantes de Grasshopper en términos de interfaz de usuario o herramientas. Se presupone un conocimiento básico de la interfaz y del flujo de trabajo. Para obtener más recursos y tutoriales de iniciación, vaya a la sección *Formación* en la web www.rhino3d.com.

El contenido se divide en tres capítulos. El capítulo 1 trata sobre los algoritmos y los datos. Introduce una metodología rigurosa para ayudar a crear y gestionar soluciones paramétricas. También presenta conceptos básicos sobre los datos, como los tipos de datos, las fuentes y las formas habituales de procesarlos. El capítulo 2 analiza las estructuras de datos básicas en Grasshopper, incluidos los elementos individuales y las listas. El capítulo 3 revisa en profundidad la estructura de datos de árbol en Grasshopper y las aplicaciones prácticas en problemas de diseño. Todos los ejemplos y tutoriales de Grasshopper están escritos con la versión 6 de Rhinoceros y se incluyen en la descarga.

Rajaa Issa

Robert McNeel & Associates

Capítulo 1: Algoritmos y datos

Los algoritmos y los datos son las dos partes esenciales de cualquier solución de diseño paramétrico, pero escribir algoritmos no es trivial y requiere una habilidad que no resulta fácil para los diseñadores intuitivos. El proceso de diseño algorítmico es muy lógico y requiere una declaración explícita de la intención de diseño y de los pasos para lograrla. Este capítulo presenta una metodología para ayudar a los diseñadores creativos a desarrollar nuevas soluciones algorítmicas. Todos los algoritmos implican la manipulación de datos, por lo que los algoritmos y los datos están estrechamente relacionados. Presentaremos los conceptos básicos de tipos de datos y procesos.

1_1: Diseño algorítmico

Podemos definir el diseño algorítmico como un método de diseño en el que el **resultado** se consigue mediante **pasos bien definidos**. En ese sentido, muchas actividades humanas son algorítmicas. Por ejemplo, hacer una pastel. El **pastel** (salida) se hace utilizando una **receta** (pasos bien definidos). Cualquier cambio en los **ingredientes** (entrada) o en el proceso de cocción da como resultado un pastel diferente. Analizaremos las partes de los algoritmos típicos e identificaremos una estrategia para crear soluciones algorítmicas desde cero.

Independientemente de su complejidad, todas las soluciones algorítmicas tienen 3 bloques de construcción: **entrada** (*input*), **proceso clave** (*key process*) y **salida** (*output*). Tenga en cuenta que el proceso clave puede requerir entradas y procesos adicionales.

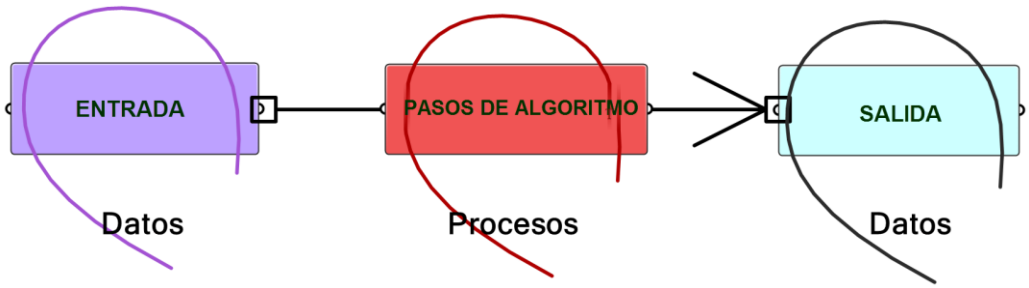


Figura (1): Bloques de construcción de soluciones algorítmicas

A lo largo de este texto, organizaremos y etiquetaremos las soluciones para identificar claramente los tres bloques. También utilizaremos un código de colores coherente para distinguir visualmente las partes. Así podremos leer más cómodamente los algoritmos, identificar rápidamente la entrada y los pasos clave del procesamiento, y obtener y mostrar correctamente la salida. Las señales visuales son importantes para desarrollar fluidez en el pensamiento algorítmico.

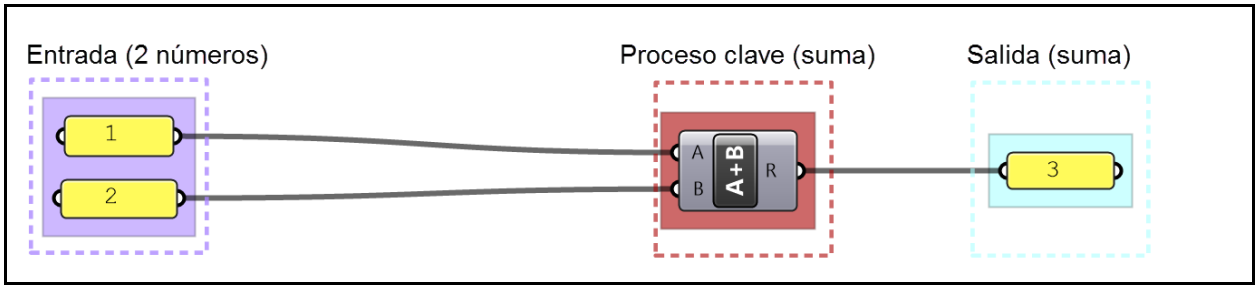
En general, leer las soluciones algorítmicas existentes es relativamente fácil, pero crear nuevas desde cero es mucho más difícil y requiere nuevas habilidades. Aunque es útil saber leer y modificar las soluciones existentes, es esencial desarrollar habilidades de diseño algorítmico para crear nuevas soluciones desde cero.

1_2: Partes de algoritmos

En Grasshopper, una solución discurre de izquierda a derecha. En el extremo izquierdo están los valores de entrada y los parámetros, y en el extremo derecho está la salida. En medio hay uno o más procesos clave y, a veces, entradas y salidas adicionales. Tomemos un ejemplo sencillo para ayudar a identificar las tres partes de cualquier algoritmo (entrada, proceso clave, salida). El algoritmo de suma simple incluye dos números (entrada), la suma (salida) y un proceso clave que toma los números y da el resultado. Utilizaremos el color morado para la entrada, el granate para los procesos clave y el azul claro para la salida. También agruparemos y etiquetaremos las diferentes partes y organizaremos las soluciones de Grasshopper de izquierda a derecha.

Ejemplo 1-2-1:

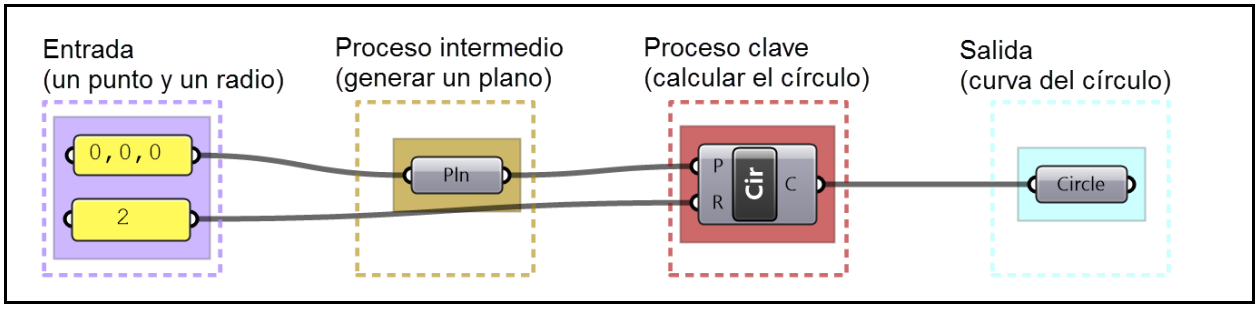
Algoritmo para sumar 2 números



Los algoritmos pueden implicar procesos intermedios. Por ejemplo, supongamos que necesitamos crear un círculo (salida) utilizando un centro y un radio (entrada). Observe que la entrada no es suficiente porque no conocemos el plano en el que debe crearse el círculo. En este caso, necesitamos generar información adicional, concretamente el plano del círculo. Lo denominaremos "proceso intermedio" y utilizaremos el color marrón para etiquetarlo.

Ejemplo 1-2-2:

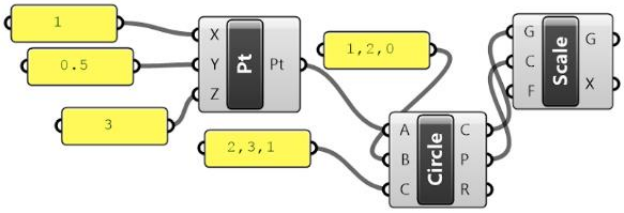
Algoritmo para crear un círculo en el plano XY a partir de un centro y un radio



Algunas soluciones no están escritas con estilos y, por lo tanto, son difíciles de leer y utilizar. Es muy importante tomarse el tiempo necesario para organizar y etiquetar sus soluciones para que sean más fáciles de entender, depurar y utilizar.

Tutorial 1-2-3: Leer un algoritmo existente

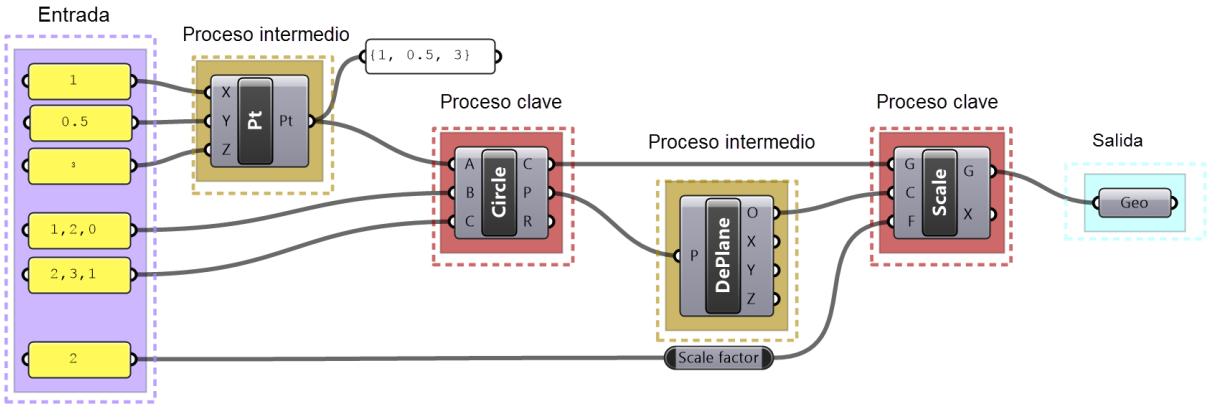
Dada la siguiente definición, escriba una descripción de lo que hace el algoritmo, identifique la entrada, el/los proceso(s) principal(es) y la salida, y luego etiquete y asigne un color a todas las partes. Reescriba la solución para hacerla más legible.



Solución

Para saber qué debe hacer el algoritmo, tenemos que agrupar la entrada en el lado izquierdo, obtener la salida en el lado derecho y luego organizar los procesos en orden de ejecución. A continuación, recorremos la solución de izquierda a derecha para deducir lo que hace. Podemos examinar y previsualizar la salida en cada paso.

El ejemplo del tutorial pretende crear un círculo que sea el doble de grande que otro círculo que pasa por tres puntos dados. Uno de los puntos se crea a partir de sus 3 coordenadas.



1_3: Diseño de algoritmos: el proceso de 4 pasos

Antes de generalizar un método para diseñar algoritmos, examinemos un algoritmo que utilizamos habitualmente en la vida real, como cocinar un pastel. Si ya tiene la receta de un pastel, solo necesita tener los ingredientes recomendados, mezclarlos, verterlos en un molde, ponerlo en el horno precalentado durante un tiempo determinado y, finalmente, servirlo. Si la receta está bien documentada, es relativamente fácil de utilizar. A medida que adquiera más práctica en la elaboración de pasteles, podrá empezar a modificar la receta. Tal vez añada nuevos ingredientes (chocolate o frutos secos) o utilice diferentes herramientas (molde para magdalenas).

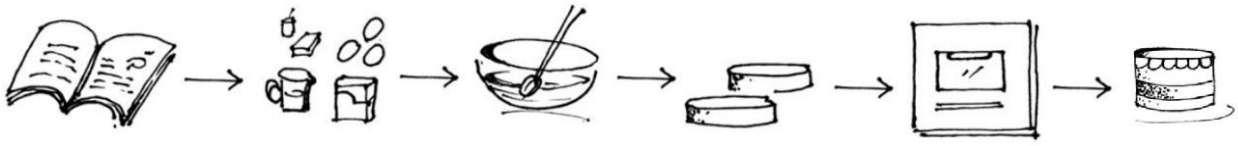


Figura (2): Pasos para seguir una receta existente

Cuando los diseñadores escriben algoritmos, normalmente intentan buscar soluciones existentes y modificarlas para que se ajusten a sus propósitos. Aunque este es un buen punto de partida, utilizar soluciones existentes puede ser frustrante y requerir mucho tiempo. Además, las soluciones existentes tienen una esencia propia y eso puede influir en las decisiones de diseño y limitar la creatividad. Si los diseñadores tienen problemas únicos, y suelen tenerlos, no les queda más remedio que crear nuevas soluciones desde cero, aunque es una tarea mucho más difícil.

Volviendo a nuestro ejemplo, la tarea de cocinar un pastel es mucho más difícil si no se tiene una receta y si no se ha hecho uno antes. Se tienen que adivinar los ingredientes y el proceso. Lo más probable es que en los primeros intentos se obtengan malos resultados, hasta descubrir cómo se hace. En general, cuando se crea una nueva receta, hay que seguir el proceso a la inversa. Se empieza con una imagen del pastel deseado y se adivinan los ingredientes, los utensilios y los pasos. El pensamiento va en la siguiente línea:

- El pastel se tiene que cocinar, así que necesito un **horno** y **tiempo**.
- Lo que va en el horno es una **masa de tarta** dentro de un **recipiente**.
- La masa es una mezcla de **ingredientes**.

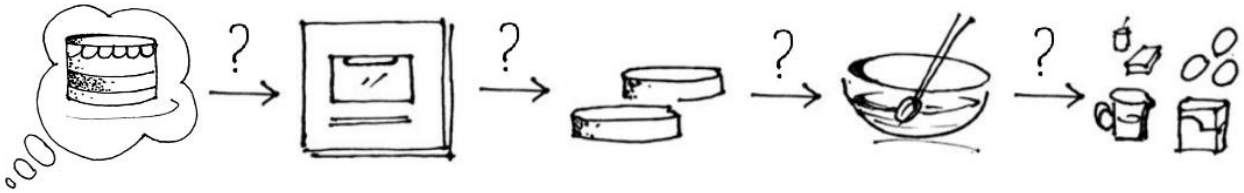


Figura (3): Pasos para inventar una nueva receta desde cero

Podemos utilizar una metodología similar para diseñar algoritmos paramétricos desde cero. Tenga en cuenta que la creación de nuevos algoritmos es una "habilidad" y requiere paciencia, práctica y tiempo para desarrollarla.

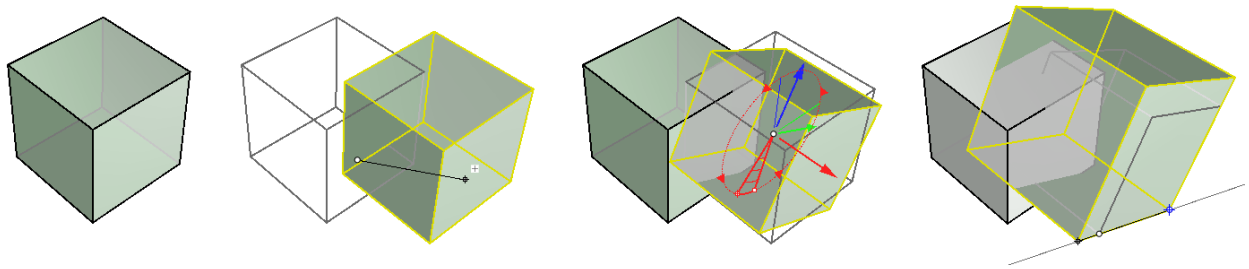
Pensamiento algorítmico en el modelado 3D frente al diseño paramétrico

El modelado 3D implica un cierto nivel de pensamiento algorítmico, pero tiene muchos datos y pasos implícitos. Por ejemplo, diseñar un modelo de masa utilizando un modelador 3D puede implicar los siguientes pasos:

1- Pensar en la salida (por ejemplo, una masa formada por algunas cajas intersectadas).
2- Identificar un comando o una serie de comandos para lograr el resultado (por ejemplo, ejecutar el comando ***Caja*** unas cuantas veces, ***Mover***, ***Escalar*** o ***Rotar*** una o más cajas, y después aplicar una ***UniónBooleana*** a la geometría).

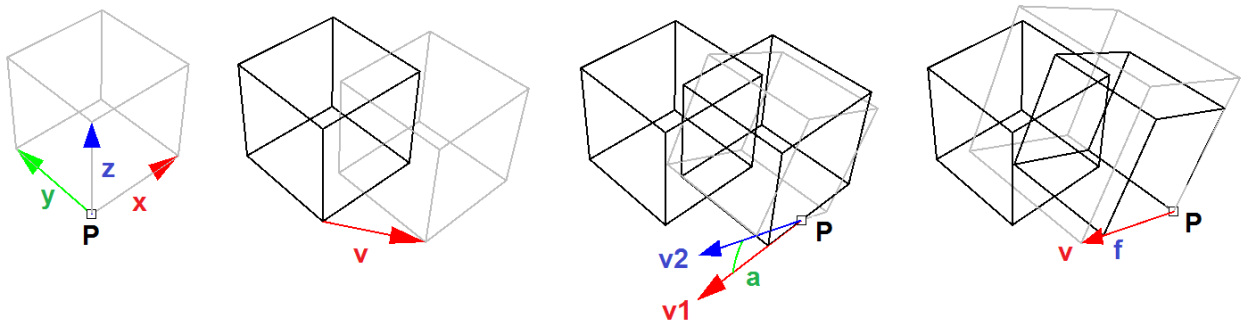
¡Ya lo tiene!

Los comandos van pidiendo datos como el punto base para la caja inicial, la anchura, la altura, el factor de escala, la dirección de movimiento, el ángulo de rotación, etc., y el diseñador no necesita prepararlos con antelación. Además, el resultado final (la masa booleana) pasa a estar directamente disponible y visible como objeto en el documento.



Figura(4): El modelado 3D interactivo para crear y manipular la geometría utiliza widgets y guías visuales

Las soluciones algorítmicas no son interactivas y requieren una conexión explícita de los datos y los procesos. En el ejemplo de la caja, hay que definir la orientación y las dimensiones de la caja. Al copiar, es necesario definir un vector y al rotar es necesario definir el plano y el ángulo o la rotación.



Figura(5): Las soluciones algorítmicas implican la definición explícita de la geometría, los vectores y las transformaciones

Diseño de algoritmos

El diseño de algoritmos requiere conocimientos de geometría, matemáticas y programación. Los conocimientos de geometría y matemáticas se recogen en el manual ***Matemáticas Esenciales para Diseño Computacional***[1]. En cuanto a las habilidades de programación, se necesita tiempo y práctica para tener la capacidad de formular las intenciones de diseño en pasos lógicos para procesar y gestionar los datos geométricos. Para empezar, piense en cualquier algoritmo como un proceso de 4 pasos, como se muestra a continuación:

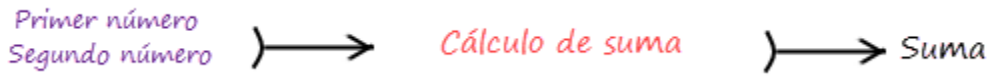
1- Identificar claramente el resultado deseado	**Salida**
2- Identificar los pasos clave para lograr el resultado	**Procesos clave**
3- Examinar los parámetros y los datos iniciales	**Entrada**
4- Definir los pasos intermedios para generar los datos que faltan	**Entrada intermedia + procesos**

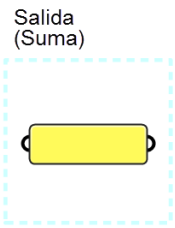
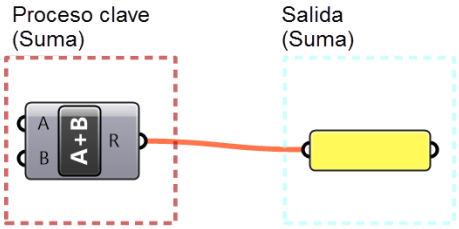
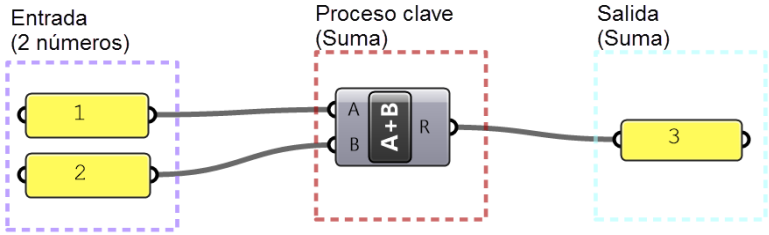
Tener en cuenta este procedimiento de 4 pasos es clave para desarrollar la habilidad del diseño algorítmico. Empezaremos con ejemplos sencillos para ilustrar la metodología, y poco a poco iremos aplicando ejemplos más complejos.

[1] Issa, Matemáticas Esenciales para Diseño Computacional, 4ª edición, 2019. Descarga gratuita del PDF y de los ejemplos: https://www.rhino3d.com/download/rhino/6/essentialmathematics

Ejemplo 1-3-1: Sumar dos números

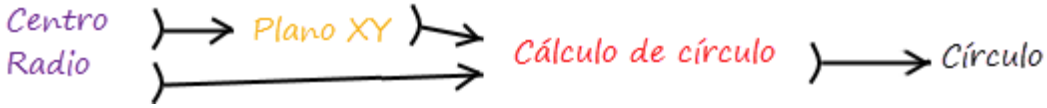
Utilice el proceso de 4 pasos para escribir un algoritmo que sume dos números

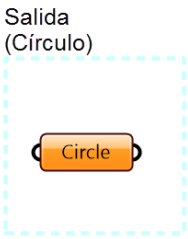


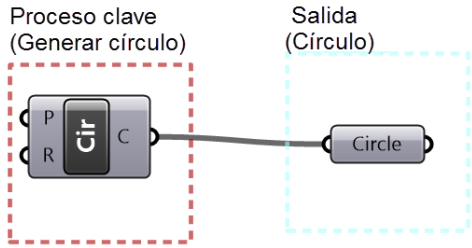
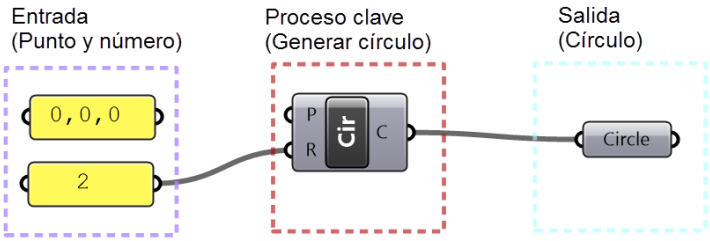
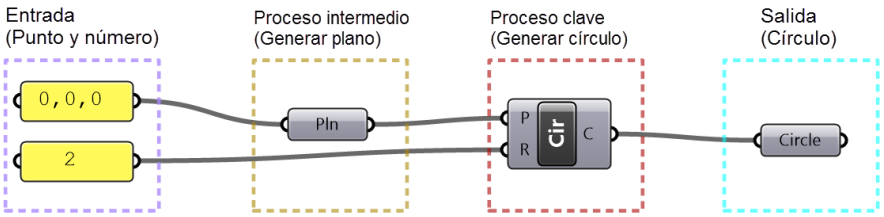
Paso 1: Salida: La suma de los 2 números. Utilice un ***Panel*** para obtener la suma.	
Paso 2: Proceso clave: Suma. Utilice el componente ***Addition*** que usa 2 números y da la suma.	
Paso 3: Entrada: 2 números. Utilice un ***Panel*** para mantener los valores de los números de entrada.	

Ejemplo 1-3-2: Crear un círculo

Utilice el proceso de 4 pasos para crear un círculo a partir de un centro y un radio

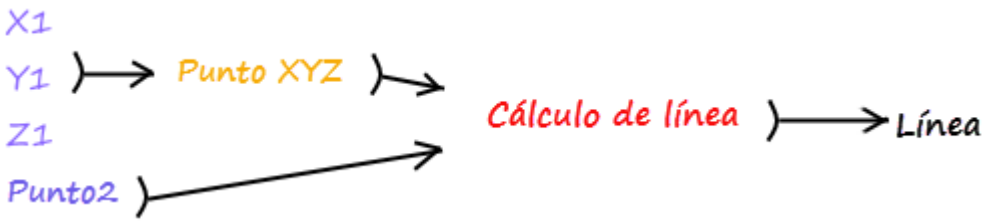


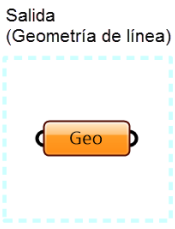
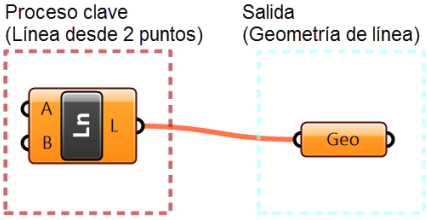
Paso 1: Salida: Círculo. Utilice el parámetro ***Circle*** para obtener la salida.	

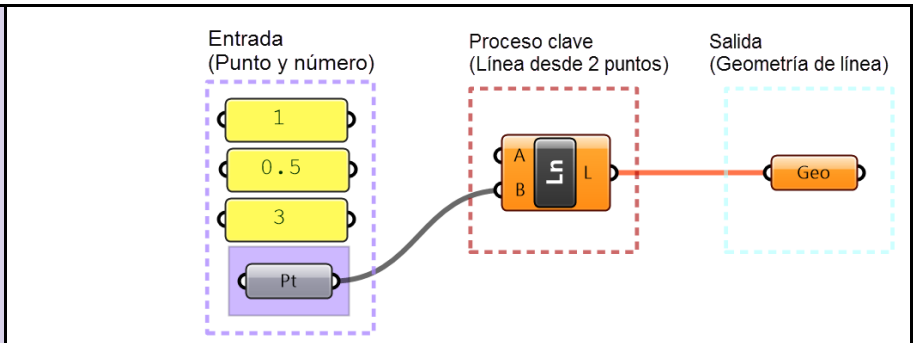
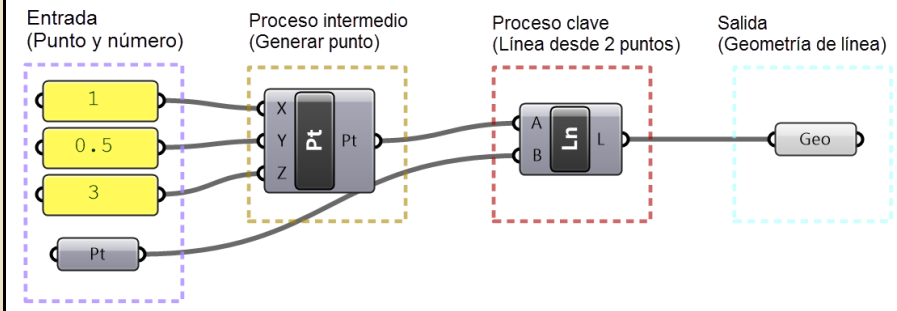
Paso 2: Proceso clave: Identifique un proceso clave que genere un círculo a partir de un radio. Utilice el componente ***Circle*** en Grasshopper.	
Paso 3: Entrada: Utiliza la entrada dada (centro y radio). Conecte el radio con el componente ***Circle***.	
Paso 4: Proceso intermedio: El círculo necesita el centro y también el plano en el que se encuentra el círculo. Considere que el círculo está en un plano paralelo al plano XY y utilice el centro del círculo como origen del plano.	

Ejemplo 1-3-3: Crear una línea

Utiliza el proceso de 4 pasos para crear un algoritmo que genere una línea a partir de 2 puntos. Un punto está referenciado desde Rhino y el otro se crea utilizando tres coordenadas (x=1, y=0,5 y z=3).



Paso 1: Salida: Geometría de la línea. Utilice el parámetro ***Geometry*** para obtener el resultado.	
Paso 2: Proceso clave: Identifique un proceso clave que genere una línea a partir de 2 puntos. Utilice el componente ***Line*** en Grasshopper.	

Paso 3: Entrada: Utilice la entrada dada (punto de referencia y 3 coordenadas). Conecte un punto con uno de los extremos de la línea.	
Paso 4: Proceso intermedio: Antes de poder utilizar las coordenadas como punto, tenemos que crear un punto.	

En los algoritmos más complejos, tendremos que analizar los problemas, investigar las posibles soluciones y desglosarlos en partes siempre que sea posible para hacerlos más manejables y legibles. En todo el manual, seguiremos utilizando el proceso de 4 pasos y otras técnicas para resolver algoritmos más complejos.

1_4: Datos

Los datos son información que se almacena en un equipo y que un programa procesa. Los datos pueden recogerse de diferentes fuentes, son de muchos tipos y se almacenan en estructuras bien definidas para poder utilizarlos con eficacia. Aunque hay muchas similitudes cuando se trata de datos en todos los lenguajes de programación, también hay algunas diferencias. Este manual explora los datos y las estructuras de datos específicas de Grasshopper.

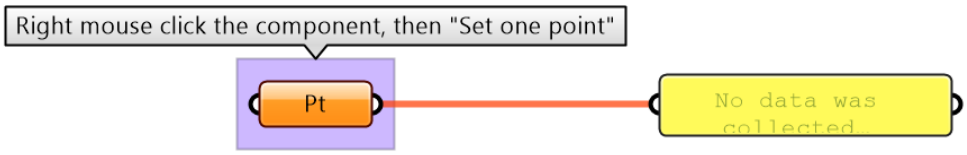
1_5: Fuentes de datos

En Grasshopper, hay tres formas principales de obtener datos para los procesos (o lo que se denominan componentes): datos internos, datos referenciados y datos externos.

Fuentes de datos en Grasshopper
1- Datos definidos internamente Los datos se pueden definir dentro de cualquier instancia de un parámetro. Una vez definidos, permanecen constantes, a menos que se modifiquen manualmente o se sobrescriban mediante una entrada externa. Este método es útil cuando no es necesario modificar los datos una vez definidos (constantes). Los datos se almacenan dentro del archivo GH.

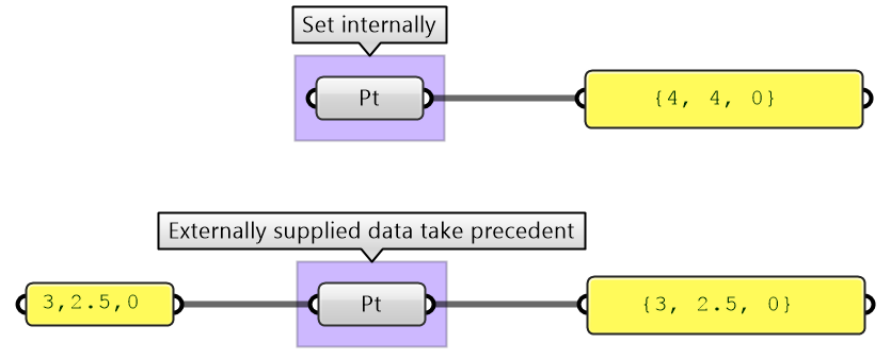
2- Datos referenciados

Los datos se pueden referenciar desde Rhino o desde un documento externo. Por ejemplo, puede referenciar un punto creado en un documento de Rhino. Cuando se mueve el punto en Rhino, su referencia en Grasshopper también se actualiza. Los archivos de Grasshopper se guardan aparte de los archivos de Rhino y, por tanto, si el archivo GH tiene datos referenciados, el archivo de Rhino debe guardarse y transferirse junto con el archivo GH para evitar pérdidas de datos.



3- Datos obtenidos externamente

Los datos se pueden obtener de procesos anteriores. Este método es el más adecuado para datos dinámicos o que se controlan paramétricamente. Los datos de un parámetro obtenidos externamente tienen prioridad sobre los datos internos o referenciados (cuando existen ambos).



1_6: Tipos de datos

Todos los lenguajes de programación identifican el tipo de datos utilizados en función de los valores que se pueden asignar y las operaciones y procesos en los que pueden participar. Existen tipos de datos comunes, como ***Integer***, ***Number***, ***Text***, ***Boolean*** (true o false) y otros. Grasshopper los enumera en la pestaña ***Params > Primitives***.

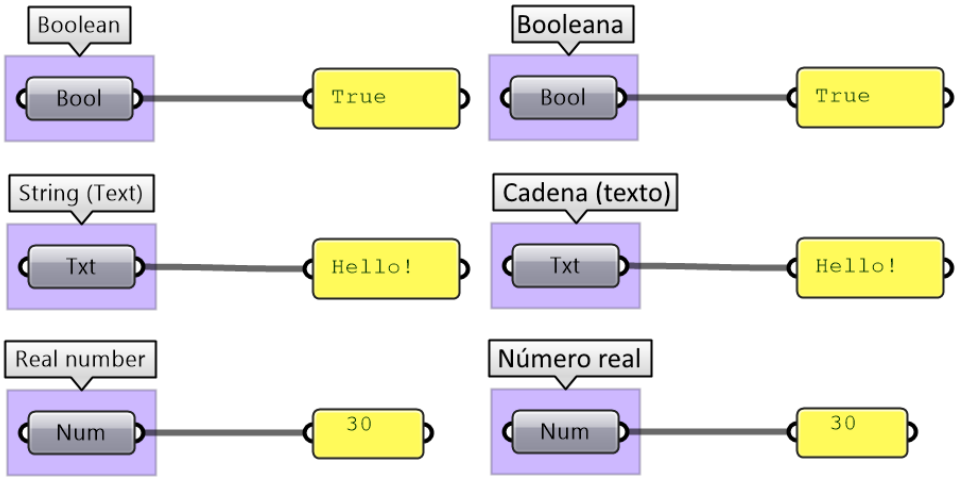


Figura (6): Ejemplos de tipos de datos primitivos comunes a todos los lenguajes de programación

Grasshopper admite tipos de geometría que son útiles en el contexto del modelado 3D, como ***Point*** (3 números para las coordenadas), ***Line*** (2 puntos), ***NURBS Curve***, ***NURBS Surface***, ***Brep*** y otros. Todos los tipos de geometría se incluyen en la pestaña ***Params > Geometry*** en Grasshopper.

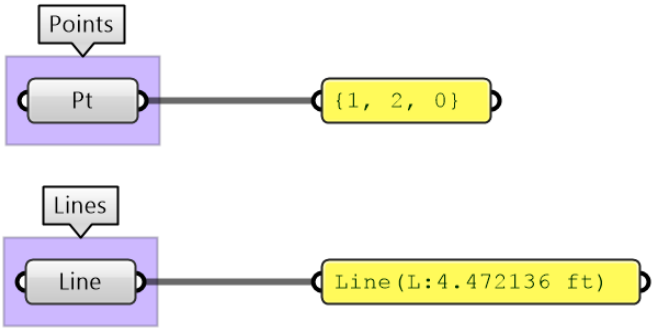


Figura (7): Ejemplos de tipos de datos geométricos

Existen otros tipos de matemáticas que los diseñadores no suelen utilizar en el modelado 3D, pero que son muy comunes en el diseño paramétrico, como ***Dominios, Vectores, Planos*** y ***Matrices de transformación***. GH ofrece un amplio conjunto de herramientas para ayudar a crear, analizar y utilizar estos tipos. Para comprender plenamente las matemáticas y los tipos de geometría como las curvas y las superficies NURBS, puede consultar el manual ***Matemáticas Esenciales para Diseño Computacional*** de la misma autora.

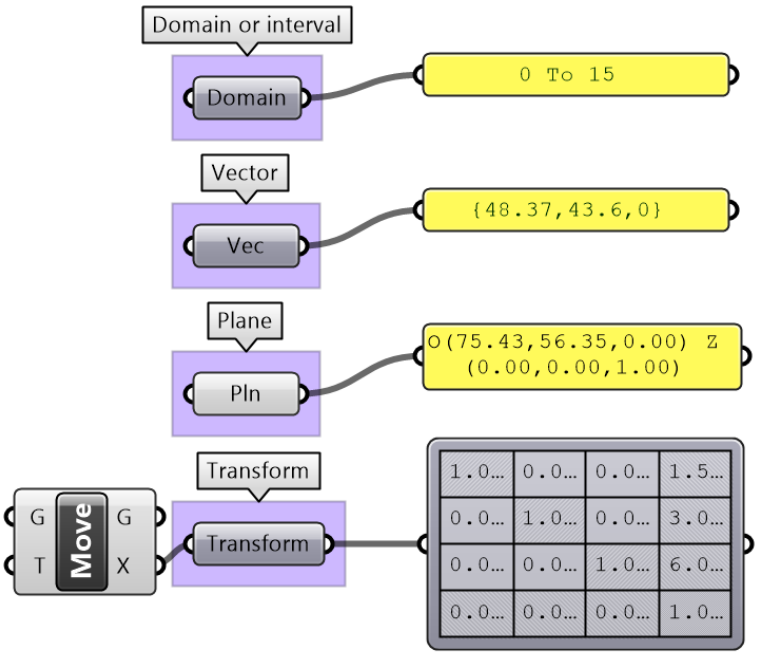


Figura (8): Ejemplos de tipos de datos habituales en diseño 3D

Los parámetros de GH pueden utilizarse para convertir datos de un tipo a otro. Por ejemplo, si quiere convertir un texto en un número, puede introducir el texto en el parámetro ***Number***. Si el texto no se puede convertir, obtendrá un error.

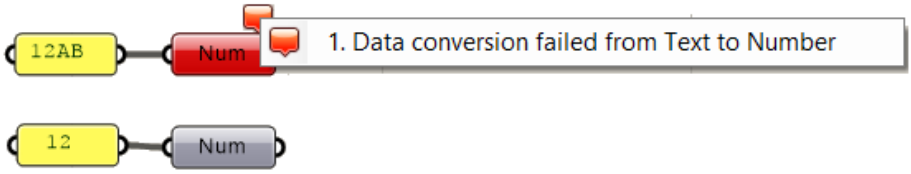


Figura (9): Conversión de datos dentro de los parámetros en Grasshopper

Los componentes de Grasshopper convierten internamente la entrada en tipos adecuados cuando es posible. Por ejemplo, si se introduce un "texto" en el componente ***Addition***, GH intenta leer el texto como un número. Si un componente puede procesar más de un tipo, utiliza el tipo de entrada sin conversión. Por ejemplo, la igualdad en una expresión puede comparar tanto texto como números. En tal caso, asegúrese de utilizar el tipo adecuado para evitar confusiones.

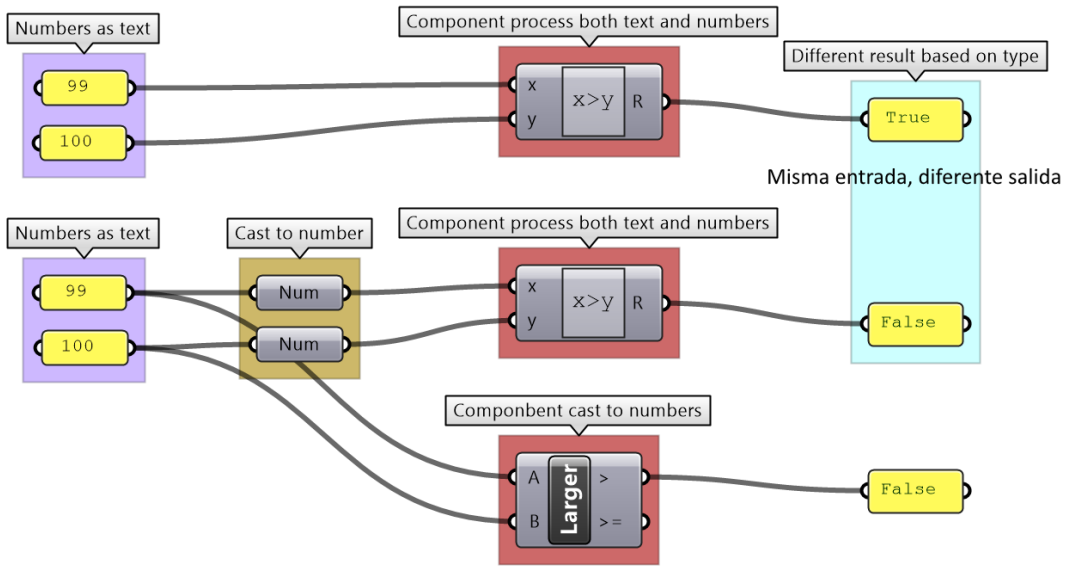


Figura (10): Algunas operaciones pueden realizarse con varios tipos. Convierta al tipo adecuado sobre todo si el componente es capaz de procesar varios tipos (como ***Expression*** en GH)

Cabe destacar que, en ocasiones, los componentes de GH simplemente ignoran las entradas no válidas (nulas o de tipo incorrecto). En estos casos, es probable que obtenga un resultado inesperado y que sea difícil encontrar el error. Es muy importante verificar la salida de cada componente antes de utilizarlo.

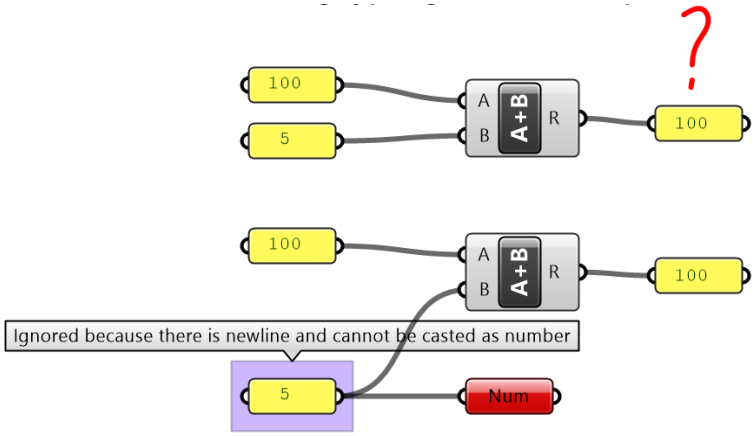


Figura (11): Las entradas no válidas se ignoran y se utiliza un valor por defecto. Por ejemplo, un número dentro de un componente ***Panel*** puede interpretarse como un texto y, por tanto, convertirse en una entrada no válida para un componente ***Addition***.

1_7: Procesamiento de datos

Los diseños algorítmicos utilizan muchas operaciones y procesos de datos. En este manual, nos centraremos en cinco categorías: operaciones numéricas y lógicas, análisis, ordenación y selección.

1_7_1: Operaciones numéricas

Las operaciones numéricas incluyen operaciones como la aritmética, la trigonometría, los polinomios y los números complejos. GH tiene un amplio conjunto de operaciones numéricas, que se encuentran principalmente en la pestaña ***Math***. Las operaciones en GH se pueden realizar principalmente de dos formas. En primer lugar, mediante el uso de componentes designados para operaciones específicas como ***Addition***, ***Subtraction*** y ***Multiplication***.

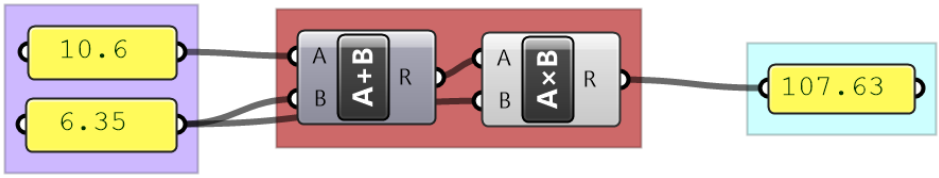


Figura (12): Ejemplos de componentes de operaciones numéricas en GH

En segundo lugar, utilice un componente de ***Expression*** donde puede combinar múltiples operaciones y realizar un amplio conjunto de operaciones matemáticas y trigonométricas, todo en una sola expresión.

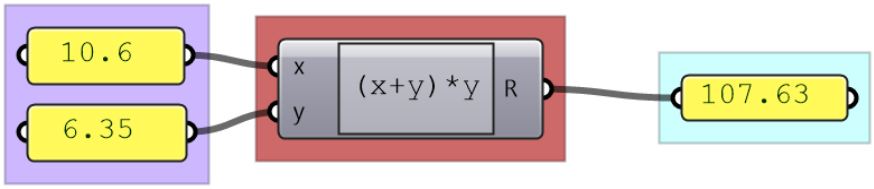


Figura (13): El componente ***Expression*** en GH se puede utilizar para realizar múltiples operaciones

El componente ***Expression*** es más sólido y legible cuando tiene múltiples operaciones.

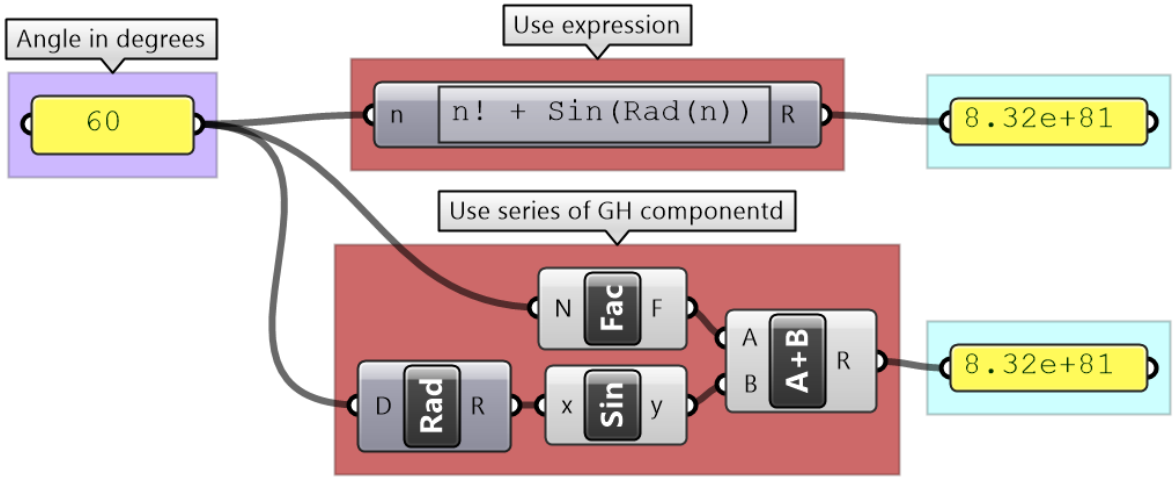


Figura (14): Cuando se trata de una cadena de operaciones, el uso del componente ***Expression*** es más fácil de mantener

La entrada a Expression se puede tratar como texto según el contexto.

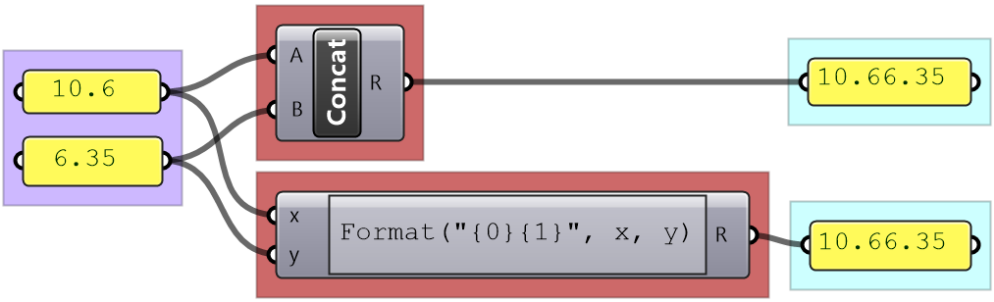


Figura (15): ***Expression*** puede procesar y formatear texto

Cabe mencionar que la mayoría de los componentes de entrada numérica permiten escribir una expresión para modificar la entrada. Por ejemplo, el componente Range tiene la entrada N (número de pasos). Si hace clic con el botón derecho del ratón en "N", puede definir una expresión. Siempre se utiliza "x" para representar la entrada proporcionada, independientemente del nombre.

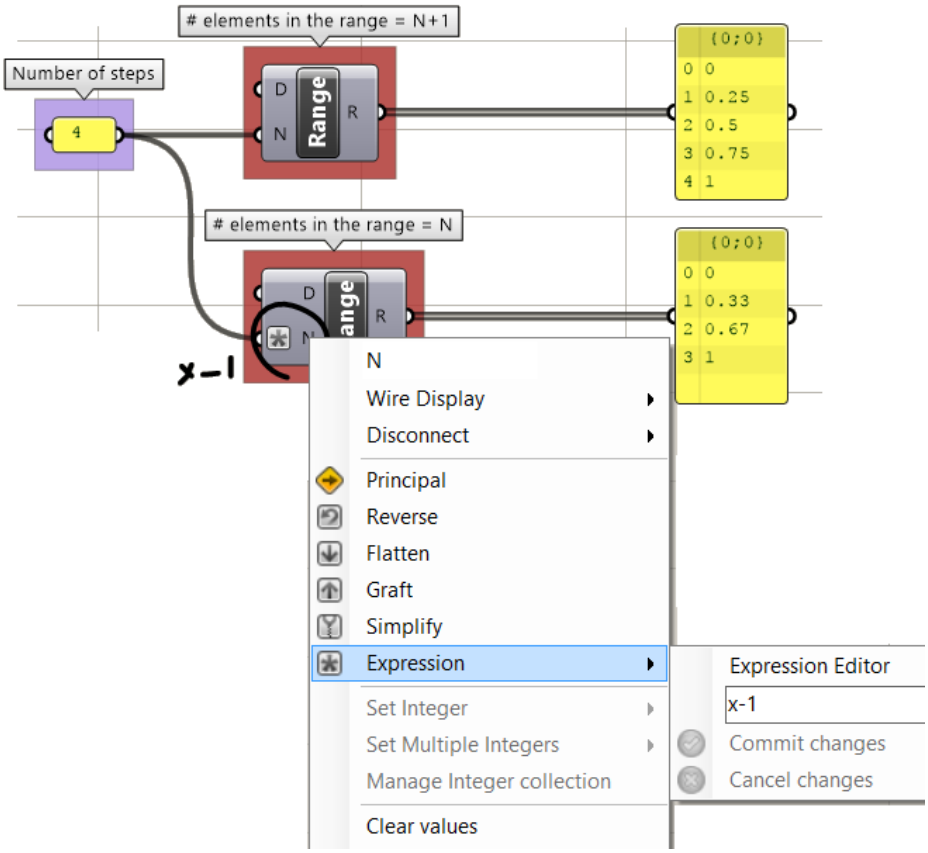


Figura (16): La expresión se puede definir dentro del parámetro de entrada. La variable "x" se refiere al valor de entrada proporcionado

1_7_2: Operaciones lógicas

Las principales operaciones lógicas en GH son las igualdades, los conjuntos y las operaciones lógicas.

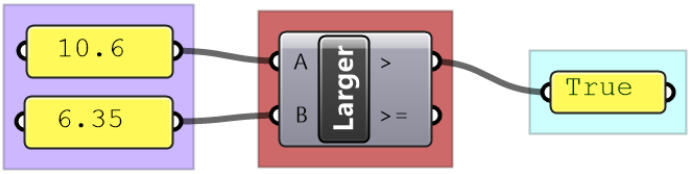


Figura (17): GH tiene varios componentes para realizar operaciones lógicas

Las operaciones lógicas se utilizan para crear un flujo de datos condicional. Por ejemplo, si quiere dibujar una esfera solo cuando el radio está entre dos valores, entonces debe crear una lógica que bloquee el radio cuando no esté dentro de los límites.

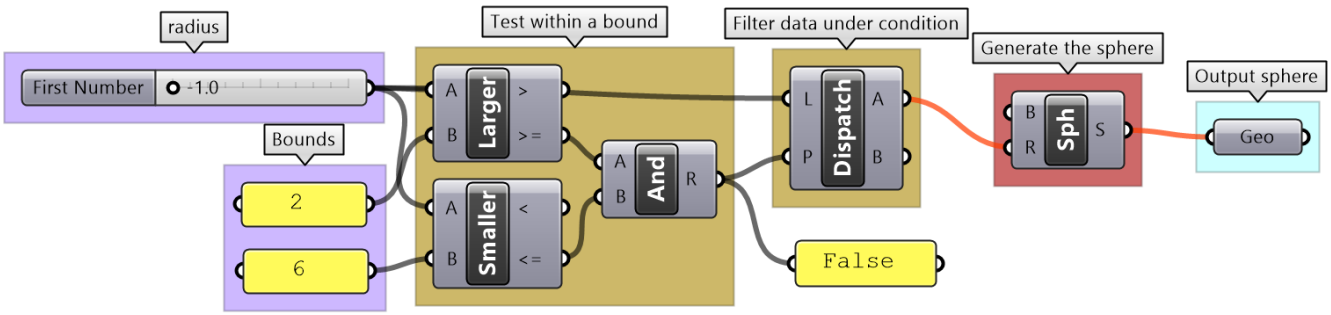


Figura (18): Control del flujo de datos mediante operaciones lógicas

1_7_3: Análisis de datos

Hay muchas herramientas en GH para examinar y previsualizar datos. ***Panel*** se utiliza para mostrar todos los detalles de los datos y su estructura, mientras que el ***Parameter Viewer*** (Visor de parámetros) muestra solo la estructura de datos. Otros componentes de análisis son el ***Quick Graph*** que traza los datos en un gráfico, y ***Bounds*** para hallar los límites en un conjunto dado de números (los valores mínimo y máximo del conjunto).

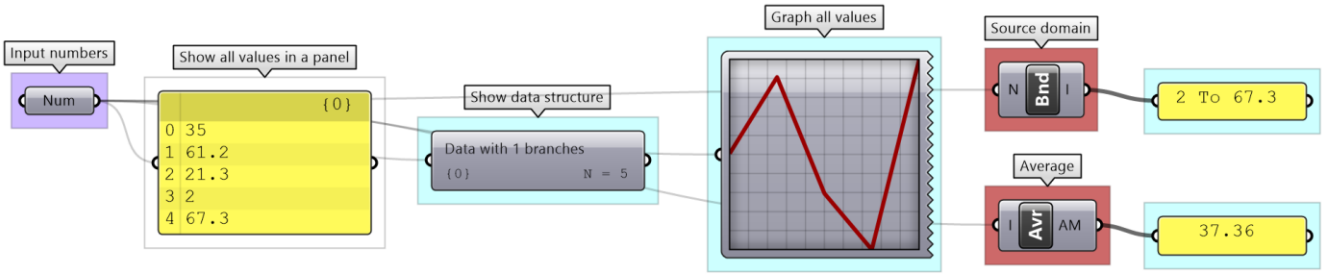


Figura (19): Algunas de las formas de analizar datos en Grasshopper

1_7_4: Ordenación

GH ha designado componentes para ordenar datos numéricos y geométricos. El componente ***Sort List*** puede ordenar una lista de claves numéricas. Puede ordenar una lista de números en orden ascendente o invertir el orden. También puede utilizar el componente ***Sort List*** para ordenar la geometría por algunas claves numéricas, por ejemplo ordenar las curvas por su longitud. GH incluye componentes para ordenar conjuntos geométricos, como ***Sort Points*** para ordenar puntos por sus coordenadas.

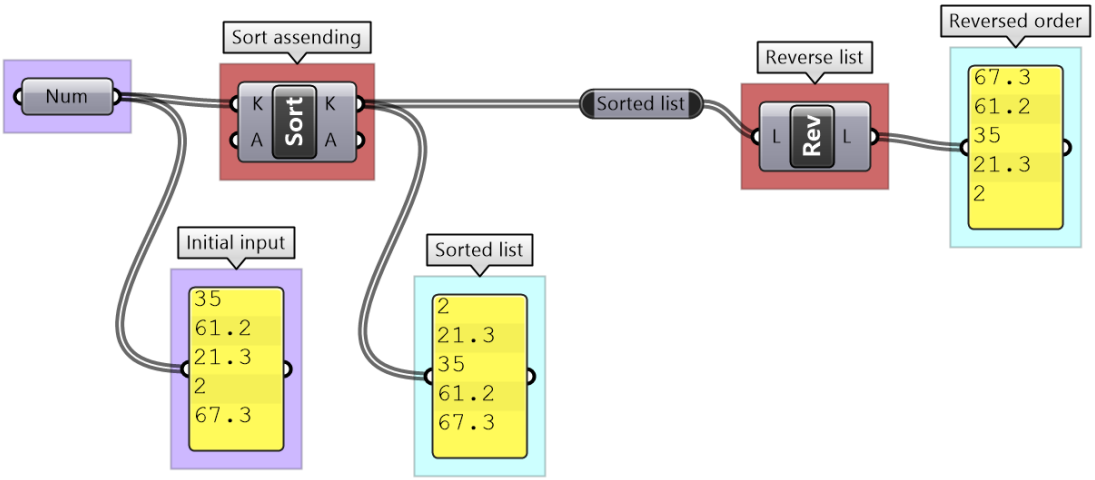


Figura (20): Ordenación de números en Grasshopper

1_7_5: Selección

El modelado en 3D permite designar objetos o grupos de objetos específicos interactivamente, pero esto no es posible con el diseño algorítmico. Los datos se seleccionan en GH en función de la ubicación dentro de la estructura de datos o mediante un patrón de selección. Por ejemplo, el componente ***List Item*** permite seleccionar elementos en función de sus índices.

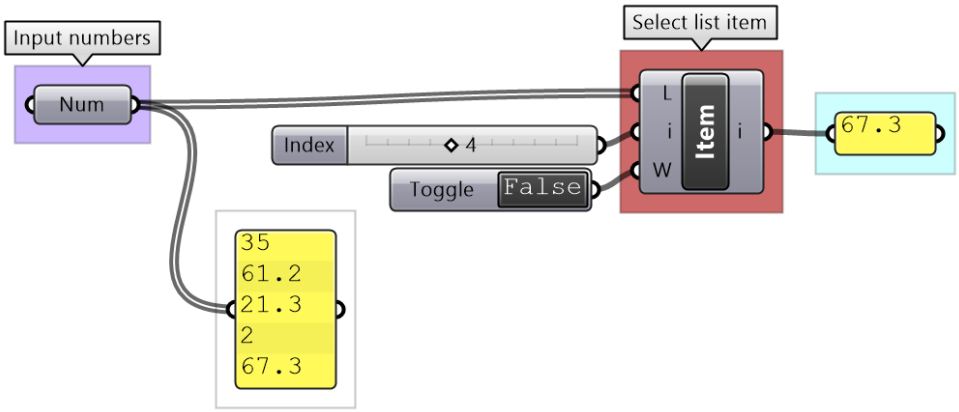


Figura (21): Selección de elementos de una lista en Grasshopper

El componente ***Cull Pattern*** permite utilizar algún patrón repetido para seleccionar un subconjunto de los datos.

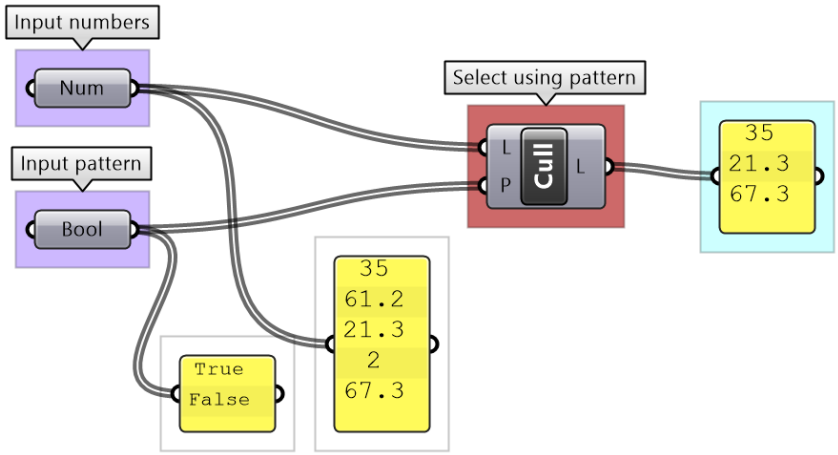


Figura (22): Ejemplo para seleccionar cualquier otro elemento de una lista

Como se puede ver en los ejemplos, la selección de elementos específicos o el uso de componentes "cull" producen un subconjunto de datos, y el resto se descarta. Muchas veces solo hay que aislar un subconjunto con el que operar y luego recombinarlo con el conjunto original. Es posible hacerlo en GH, pero implica operaciones más avanzadas. Entraremos en los detalles de estas operaciones cuando hablemos de las estructuras de datos avanzadas en el capítulo 3.

1_7_6: Mapeado

Se refiere a la asignación lineal de un intervalo de números en el que cada número de un conjunto se asigna exactamente a un valor del nuevo conjunto. GH tiene un componente para realizar mapeados lineales denominado ***ReMap***. Se puede utilizar para escalar un conjunto de números desde su intervalo original a uno nuevo. Sirve para escalar el intervalo a un dominio que se adapte a las necesidades y limitaciones de su algoritmo.

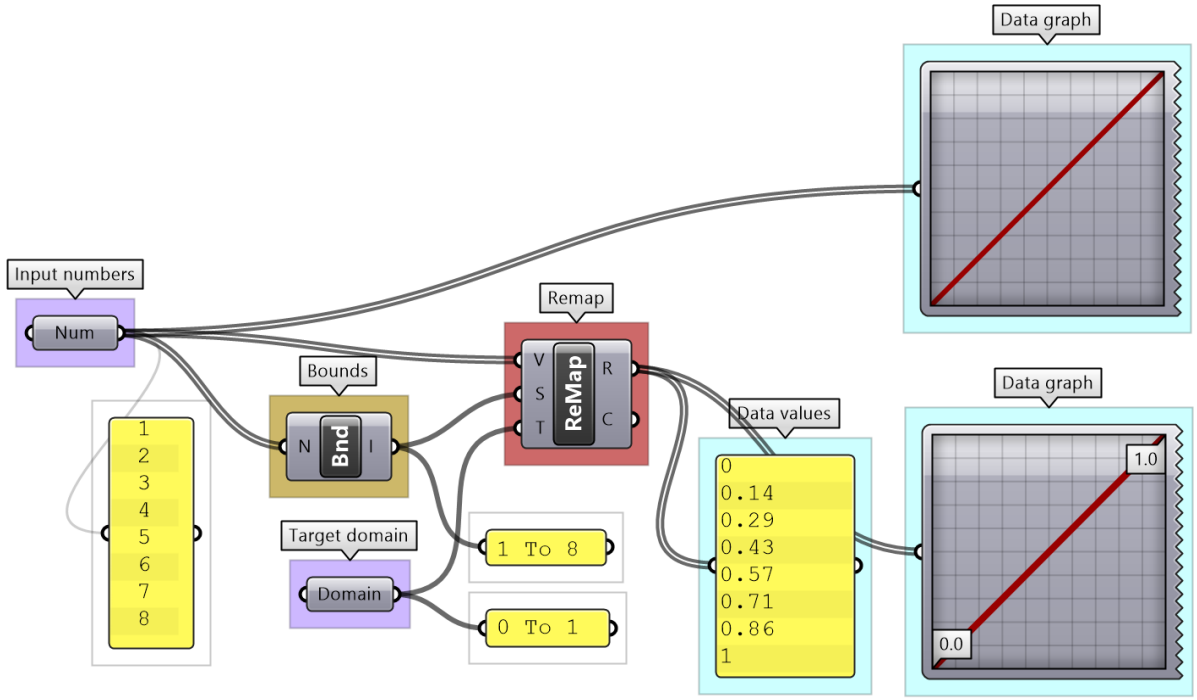


Figura (23): Ejemplo de remapeado lineal de números en Grasshopper

La conversión de los datos implica un mapeado. Por ejemplo, puede necesitar convertir una unidad de ángulo de grados a radianes (los componentes de GH solo aceptan ángulos en radianes). Por ejemplo, puede necesitar convertir una unidad de ángulo de grados a radianes (los componentes de GH solo aceptan ángulos en radianes).

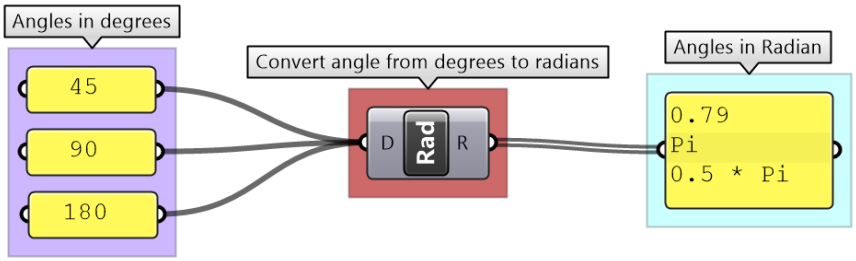


Figura (24): Convertir ángulos de grados a radianes

Como ya sabe, las curvas paramétricas tienen "dominios" (el intevalo de parámetros que se traducen a puntos en la curva). Por ejemplo, si el dominio de una curva dada está entre 12,5 y 51,3, al calcular la curva en 12,5 se obtiene el punto del inicio de la curva. Muchas veces es necesario calcular múltiples curvas utilizando parámetros coherentes. Reparametrizar el dominio de las curvas a un intervalo unificado ayuda a resolver este problema. Un dominio común que se puede utilizar es "0 To 1". En la entrada de cada curva en cualquier componente de GH, existe la opción ***Reparameterize*** , que restablece el dominio de la curva a "0 To 1".

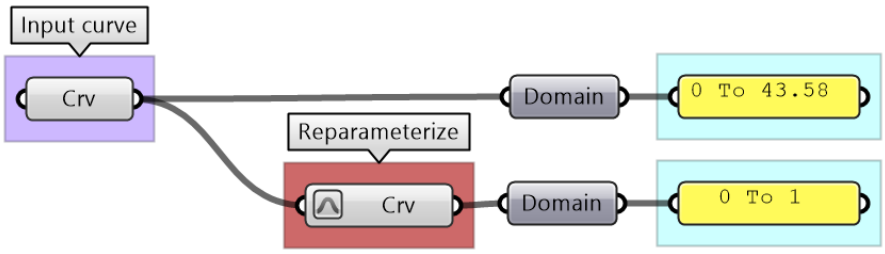
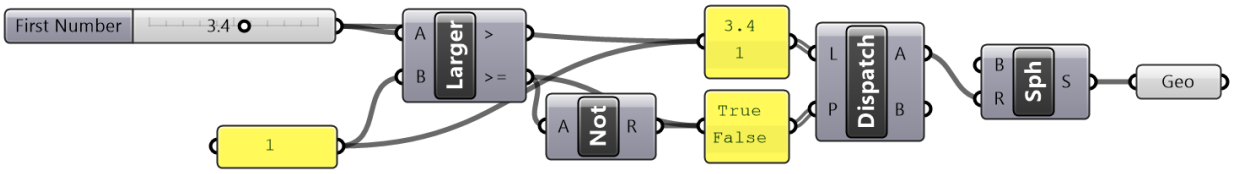


Figura (25): Normalice el dominio de las curvas (definido en 0-1). Utilice el indicador de entrada Reparameterize en Grasshopper

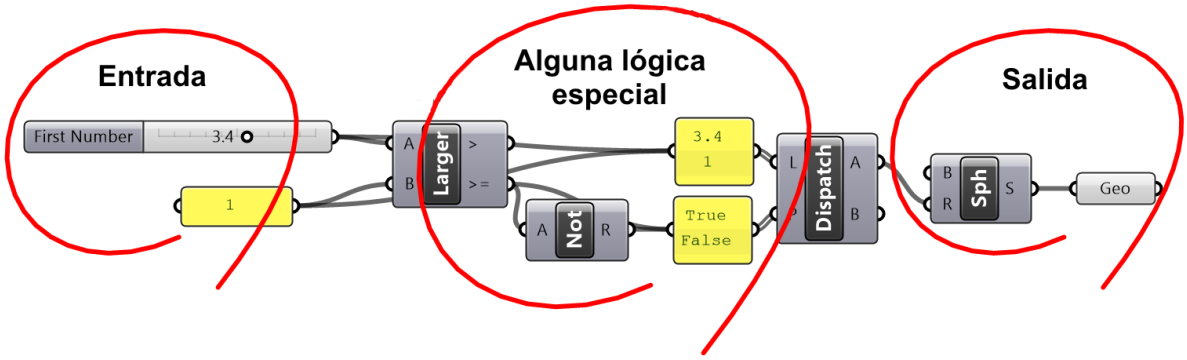
1-7-1: Tutorial de control de flujo

¿Cuál es la finalidad del siguiente algoritmo? Anote y asigne un color para describir la finalidad de cada parte.



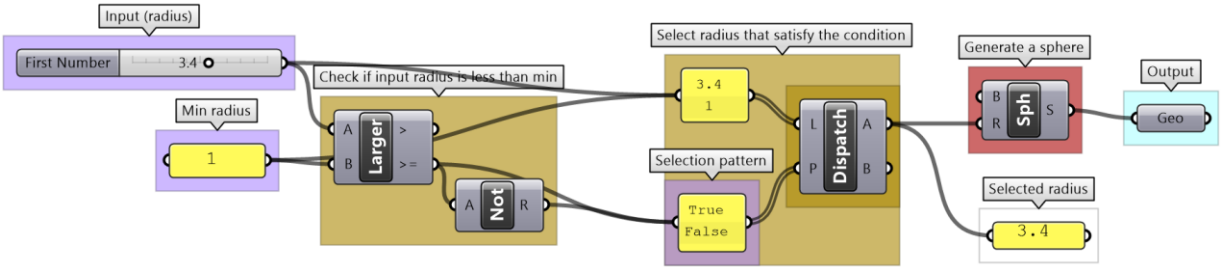
Analizar el algoritmo

El algoritmo tiene una salida que es una esfera, una entrada de radio y alguna lógica condicional para procesar el radio.

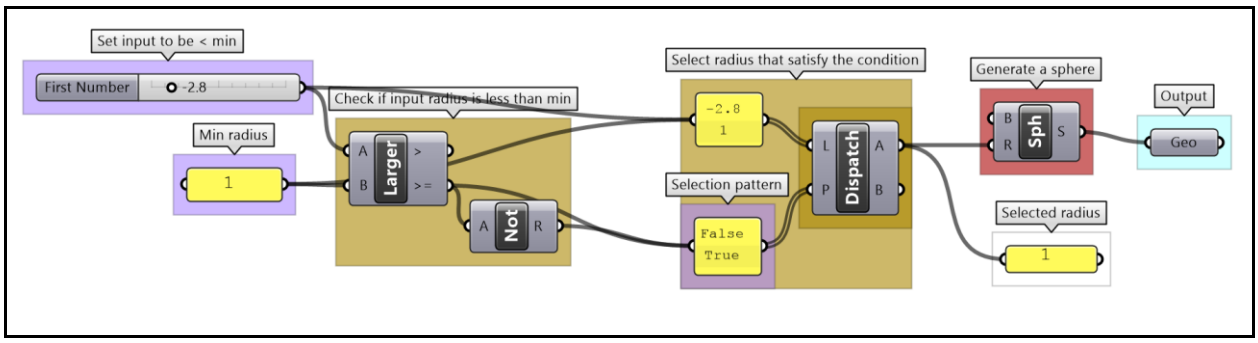


Anotar y asignar un color a la solución

Al probar la salida y seguir los pasos de la solución, parece que la intención es que el radio de la esfera no pueda ser inferior a 1 unidad.
Prueba con radio > 1

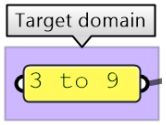
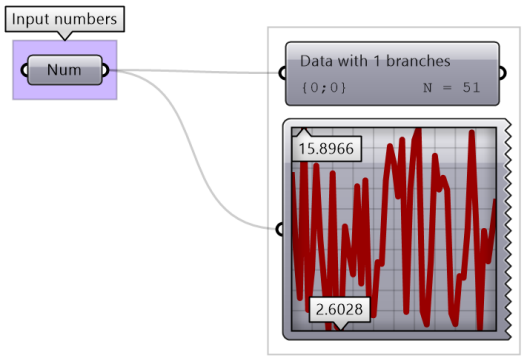
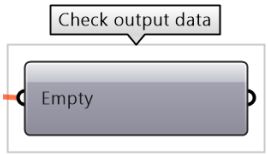
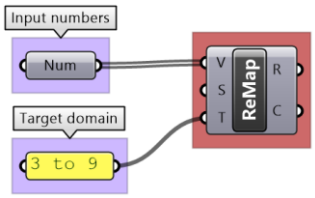


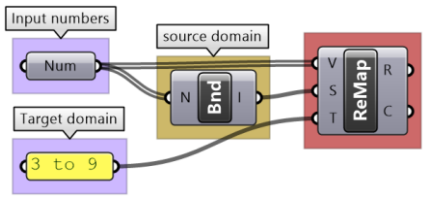
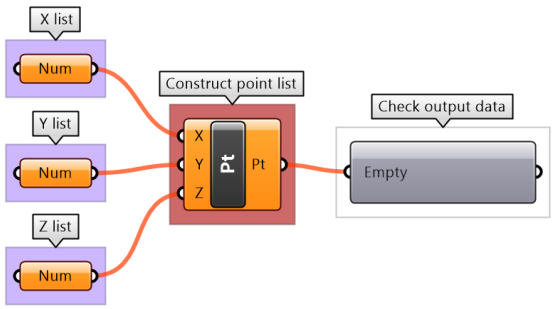
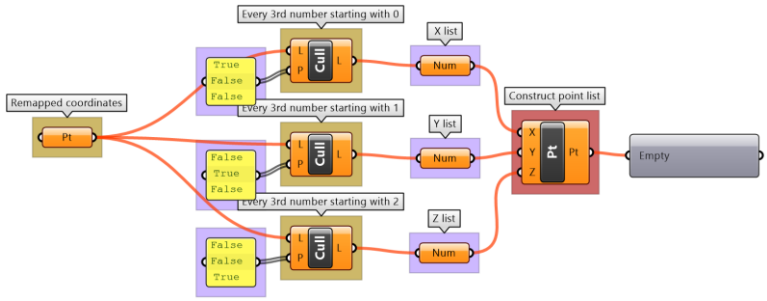
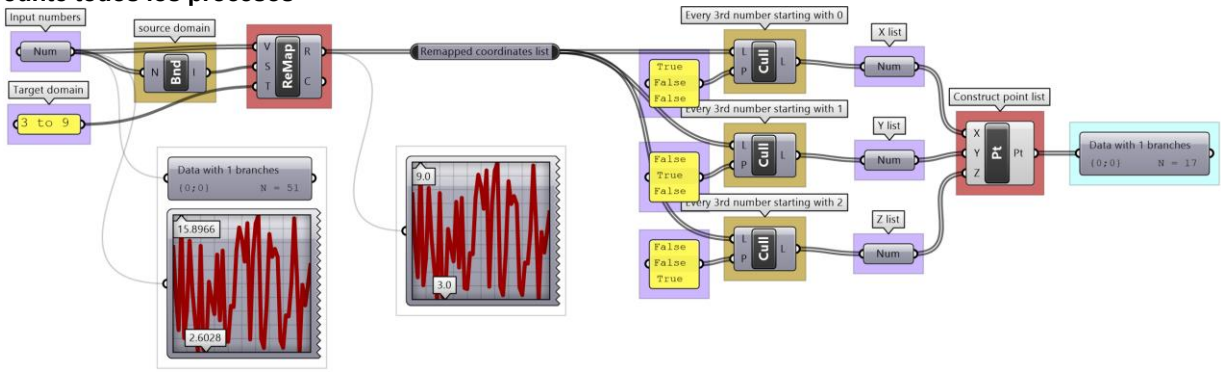
Prueba con radio < 1



1-7-2: Tutorial de procesamiento de datos

Dada una lista de coordenadas de puntos, haga lo siguiente:
1- Analice la lista para entender los datos.
2- Escriba un algoritmo que utilice la entrada para crear una lista de tipo *Point* con coordenadas asignadas a un dominio entre 3 y 9.
Observe que la lista de entrada está organizada de manera que los 3 primeros números se hacen referencia a las coordenadas x,y,z del primer punto, los 3 segundos números pertenecen al segundo punto y así sucesivamente.

Análisis del algoritmo	
Hay una lista de 51 números (3 coordenadas para cada punto implica que la lista incluye 17 puntos). Utilizando un ***QuickGraph***, podemos ver que el intervalo de valores está entre 2.60 y 15.89. También podemos ver que los valores se distribuyen aleatoriamente. Otra entrada es el dominio objetivo: 	
Utilice el proceso de 4 pasos para resolver los algoritmos	
Salida Lista de puntos	
Proceso clave 1 - Coordenadas de remapeado: Mapee la lista de coordenadas de su dominio actual a un nuevo dominio 3 a 9 Utilice el componente ***ReMap***	

Procesos intermedios 1 Falta el dominio de entrada, que puede extraerse con el componente ***Bounds***	
Proceso clave 2 - Crear puntos: Cree puntos a partir de coordenadas Utilice el componente ***Construct Point (Pt)***	
Procesos intermedios 2 Extrae todas las coordenadas X en una lista, Y en una segunda lista y Z en una tercera lista. Utilice el componente ***Cull Pattern*** con el patrón apropiado para extraer cada coordenada en una lista aparte. La entrada a ***Cull*** son los puntos remapeados del proceso 1	
Junte todos los procesos 	

1_8: Dificultades del diseño algorítmico

Escribir algoritmos elegantes que sean eficaces y fáciles de leer y depurar es difícil. En este capítulo hemos explicado cómo escribir algoritmos con estilo utilizando códigos de colores y etiquetas. También hemos planteado un proceso de 4 pasos para ayudar a desarrollar algoritmos. Seguir estas guías ayuda a minimizar los errores y a mejorar la legibilidad de los scripts. Vamos a enumerar algunos de los problemas más comunes que llevan a un resultado incorrecto o no deseado.

1_8_1: Tipo de entrada no válido o incorrecto

Si la entrada es de un tipo incorrecto o no es válida, GH cambia el color de los componentes a rojo o naranja para indicar una advertencia de error, con información sobre cuál podría ser el problema. Esto es útil, pero a veces una entrada errónea pasa desapercibida si los componentes asignan un

valor por defecto o calculan un valor alternativo para reemplazar la entrada, y eso no es lo que se pretendía. Se debería comprobar siempre la entrada (acoplarla a un panel o visor de parámetros y etiquetarla). Para evitar usar tipos incorrectos, es aconsejable convertir al tipo deseado para garantizar la precisión.

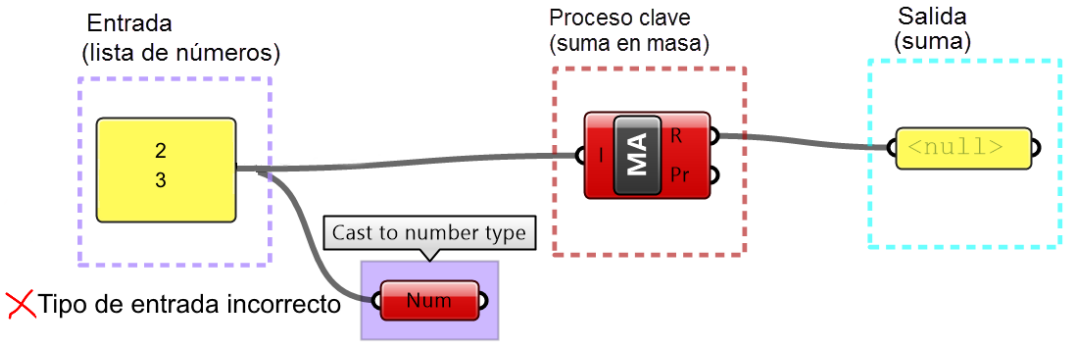


Figura (26): Error derivado de un tipo de entrada incorrecto

1_8_2: Entrada no intencionada

Las entradas pueden tener cambios no intencionados a través de procesos intermedios o cuando varios usuarios tienen acceso de escritura al script. Es muy útil previsualizar y verificar todas las entradas y salidas. El componente ***Panel*** es muy versátil y puede ayudar a comprobar todos los tipos de valores. También se puede establecer una lógica de protección contra valores fuera de rango o para detectar valores no deseados.

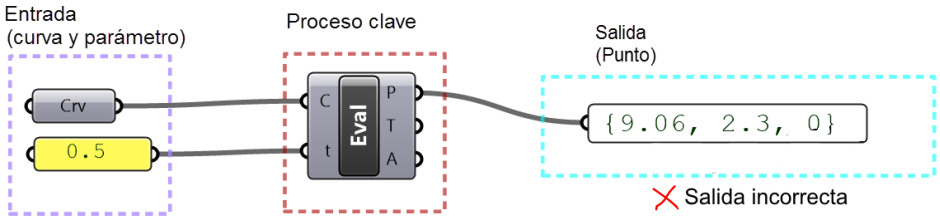


Figura (27): Error resultante de una entrada no intencionada. No se puede dar por hecho que el dominio de la curva es 0-1 y utilizar 0.5 para calcular el punto medio.

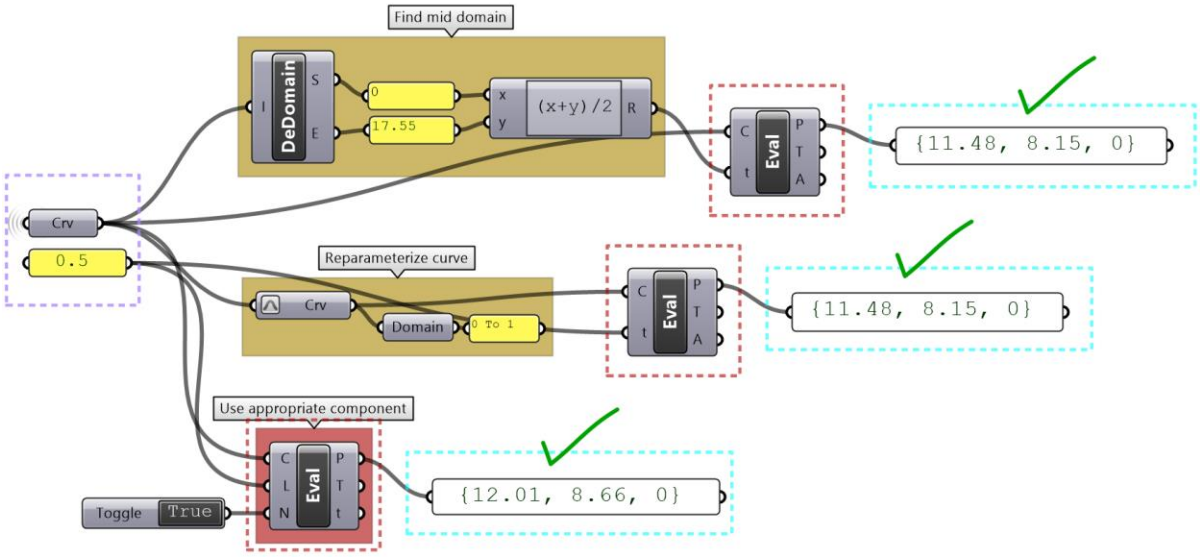


Figura (28): Ejemplo de una buena solución para calcular el punto medio de una curva

1_8_3: Orden de funcionamiento incorrecto

Hay que intentar organizar las soluciones en horizontal o en vertical para ver claramente la secuencia de operaciones. También debe comprobar la salida de cada paso para asegurarse de que es la esperada antes de continuar. También hay algunas técnicas que ayudan a consolidar el script, por ejemplo, utilizar ***Expression*** cuando hay múltiples operaciones numéricas y matemáticas involucradas. A continuación se muestran algunas organizaciones contraproducentes.

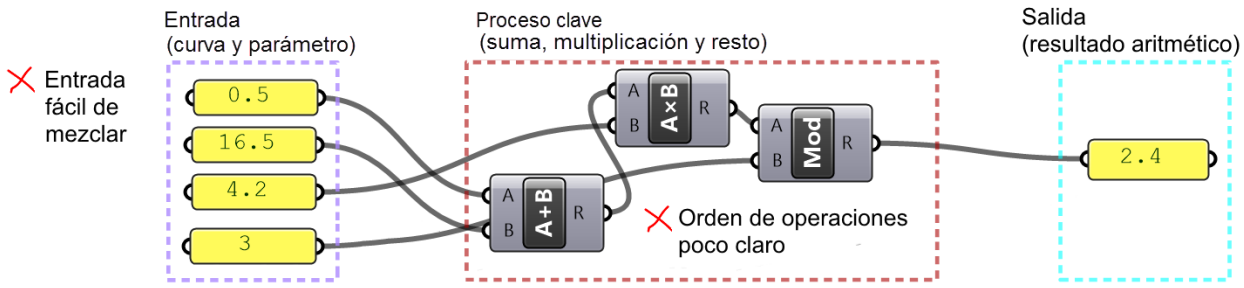


Figura (29): Es fácil confundir la entrada en operaciones con una mala organización

A continuación se muestra cómo reescribir el mismo código para que sea menos propenso a errores.

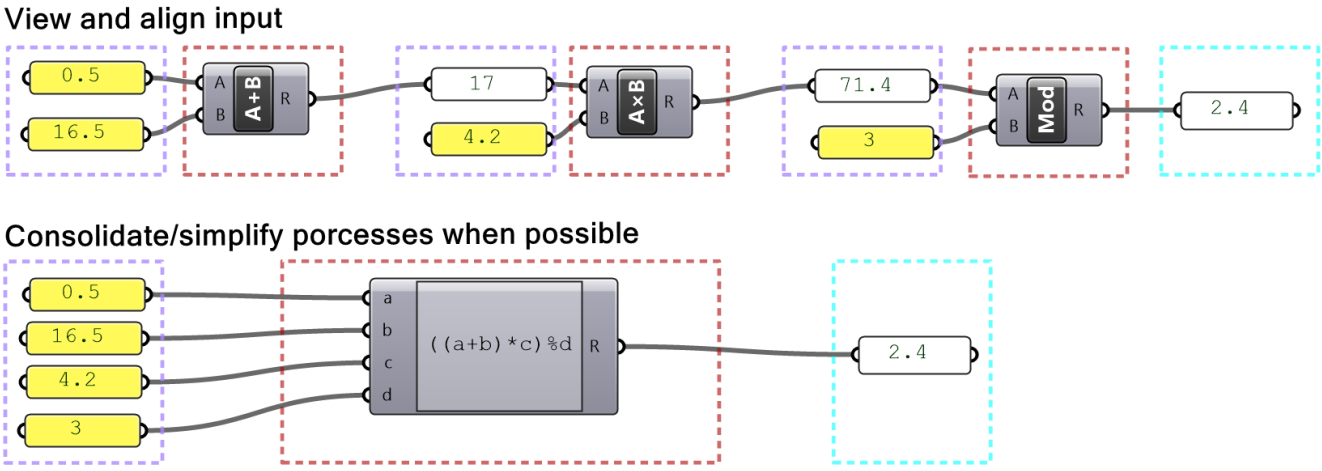


Figura (30): Mejor práctica para alinear la entrada con los procesos o utlizando ***Expressions***

1_8_4: Estructuras de datos no coincidentes

Las estructuras de datos que no coinciden como entrada en el mismo proceso o componente son particularmente difíciles de evitar en GH y pueden hacer que la solución agote la memoria. Es muy importante probar la estructura de datos de todas las entradas (excepto las triviales) antes de usarlas en cualquier componente. También es importante examinar las correspondencias deseadas en diferentes escenarios (las correspondencias de datos se explicarán con detallade más adelante).

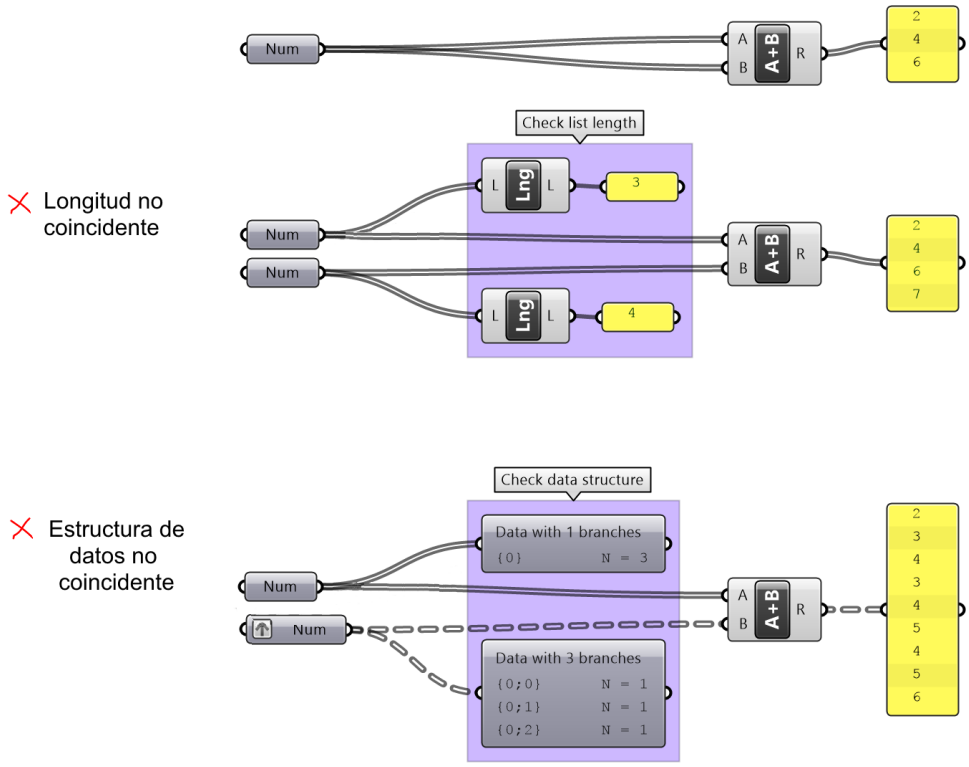


Figura (31): Las estructuras de datos de entrada no coincidentes pueden provocar errores en la salida

1_8_5: Tiempo de procesamiento largo

Algunos algoritmos consumen mucho tiempo y hay que esperar a que se procesen, pero hay formas de minimizar la espera cuando es innecesaria. Por ejemplo, en los primeros ciclos de desarrollo, utilice un conjunto de datos más pequeño para probar su solución antes de dedicar el tiempo a procesar el conjunto de datos completo. También va muy bien dividir la solución en etapas cuando siempre que posible, para poder aislar y desactivar las partes que consumen más tiempo. Además, muchas veces también se puede reescribir la solución para que esté más optimizada y consuma menos tiempo. Utilice el ***Profiler*** de GH para probar el tiempo de procesamiento. Cuando una solución tarda demasiado en procesarse o se bloquea, haga lo siguiente: antes de volver a abrir la solución, desactívela y desconecte la entrada que ha provocado el bloqueo.

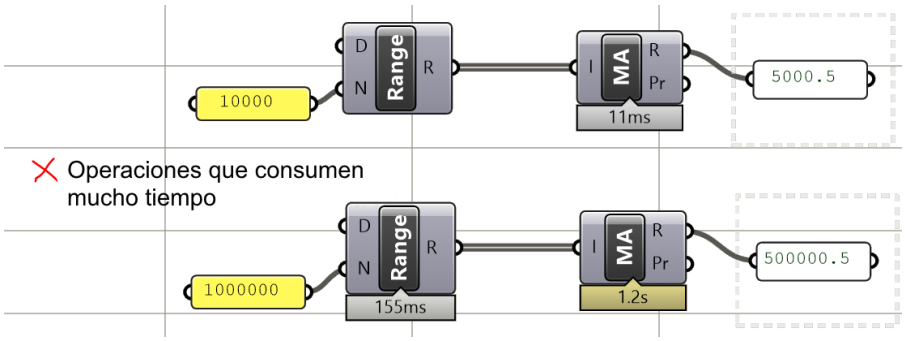


Figura (32): El widget Profiler de Grasshopper ayuda a controlar el tiempo de procesamiento

1_8_6: Mala organización

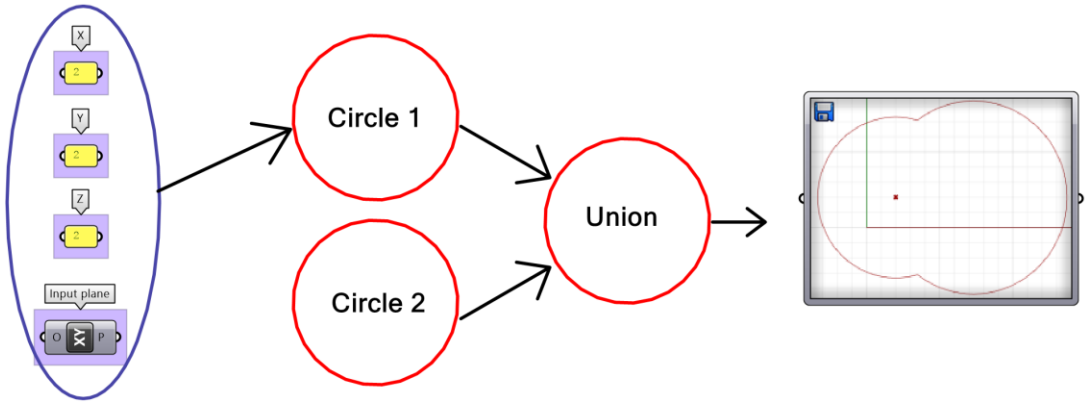
Las definiciones mal organizadas no son fáciles de depurar, comprender, reutilizar o modificar. Insistimos en la importancia de redactar las definiciones con estilos, aunque al principio cueste más tiempo. Siempre hay que codificar por colores, etiquetarlo todo, dar nombres significativos a las variables, dividir las operaciones repetidas en módulos y previsualizar la entrada y la salida.

Figura (33): Una mala organización en la programación visual hace que el código sea difícil de leer o depurar

1_9: Tutoriales de algoritmos

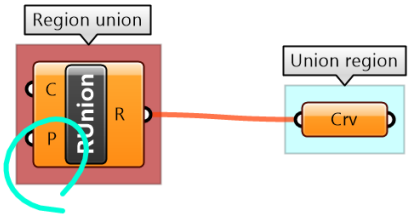
1_9_1: Tutorial de círculos unidos

Utilice el proceso de 4 pasos para diseñar un algoritmo que una 2 círculos que están situados en el plano XY. El primer círculo (Cir1) tiene centro (C1) = (2,2,2) y un radio (R1) = un número aleatorio entre 3 y 6. El segundo círculo (Cir2) tiene un centro (C2) que está desplazado a la derecha de (Cir1) una cantidad igual a R1 a lo largo del eje X positivo. R2 = R1 * 1.2

Analice la pregunta y el flujo de la solución	
	
Pasos de la solución	
Salida Curva de unión de regiones	

Proceso clave: unión de 2 círculos

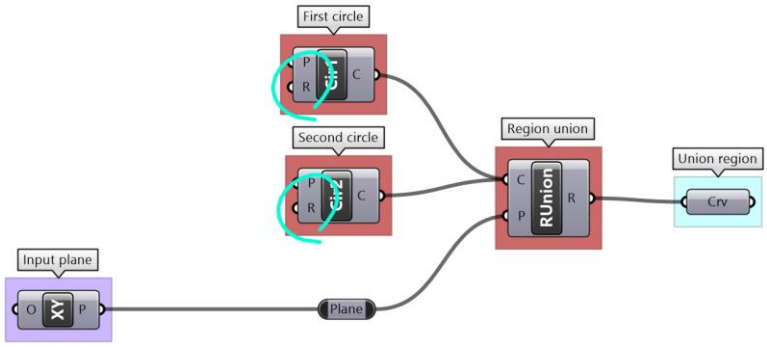
Utilice el componente ***Region Union*** que toma las curvas y un plano



Entrada a la unión de regiones

Identifique la entrada necesaria y utilícela cuando sea pertinente.

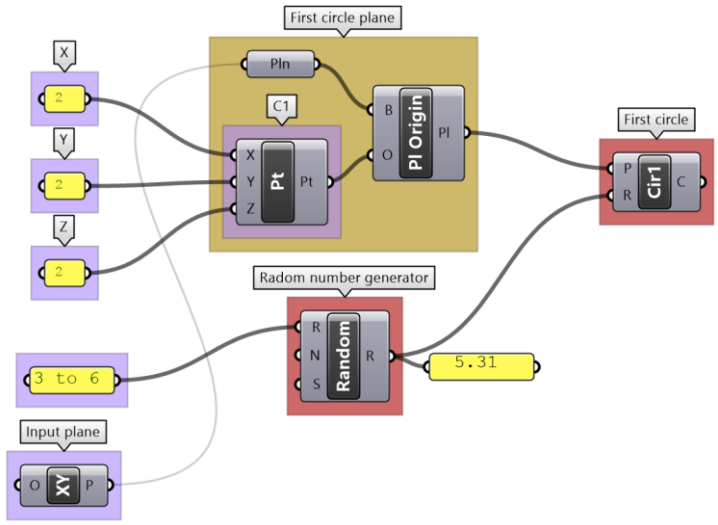
Se ha dado el plano para la unión de regiones. Los 2 círculos necesitan su propio plano y radio. El centro del plano es el centro del círculo.



Proceso intermedio para generar el centro y el plano del primer círculo

Cree un centro a partir de las coordenadas dadas. Cree un plano utilizando el componente ***Plane Origin*** y utilice el centro creado y el plano XY.

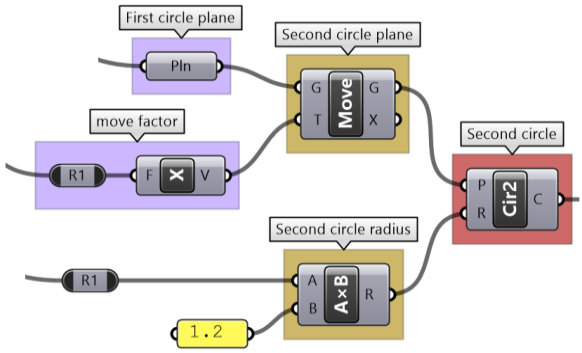
El radio es un número aleatorio entre 3 y 6. Utilice el componente ***Random*** para crear el radio.



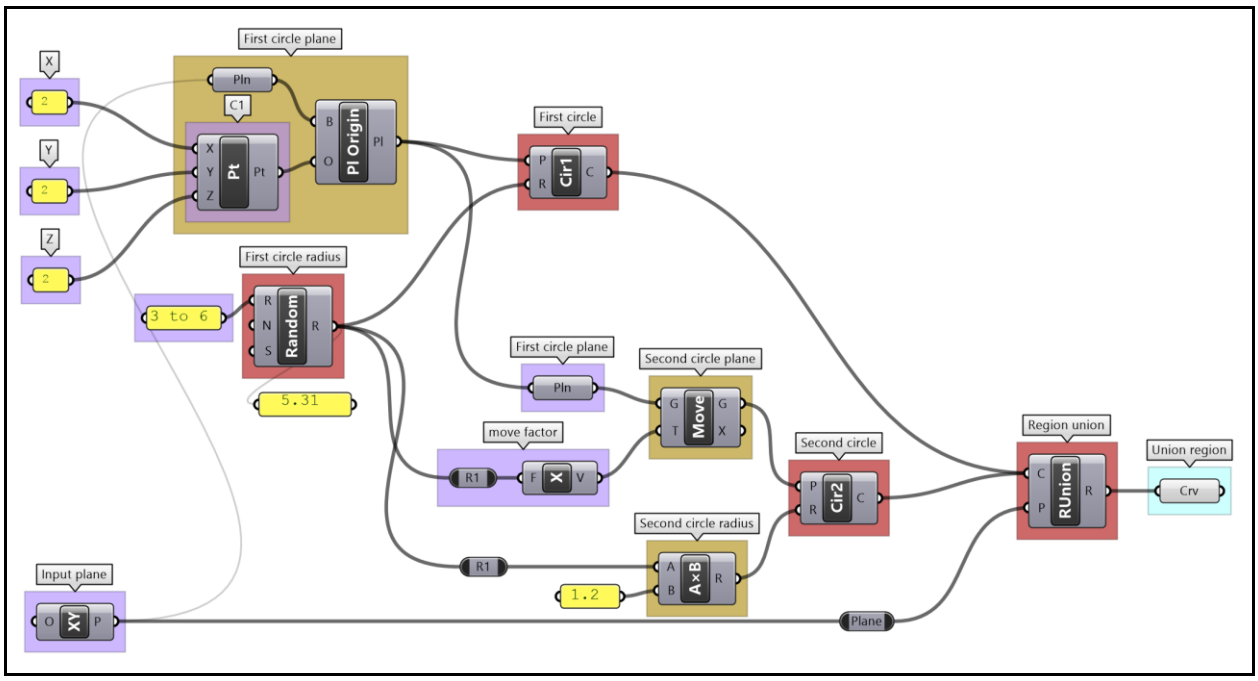
Proceso intermedio para generar el centro y el plano del segundo círculo

Calcule el segundo plano circular desplazando el primer plano circular a lo largo del eje X una cantidad igual al primer radio.

Calcule el radio del segundo círculo multiplicando el primer radio por 1.2.

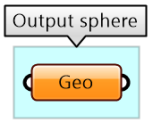
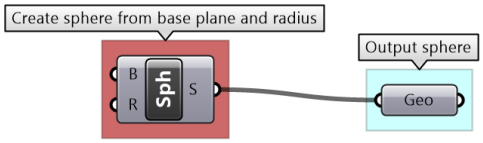
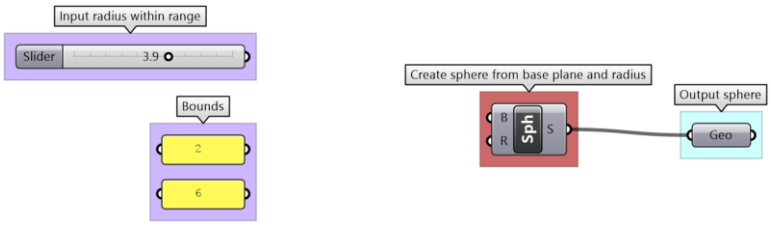


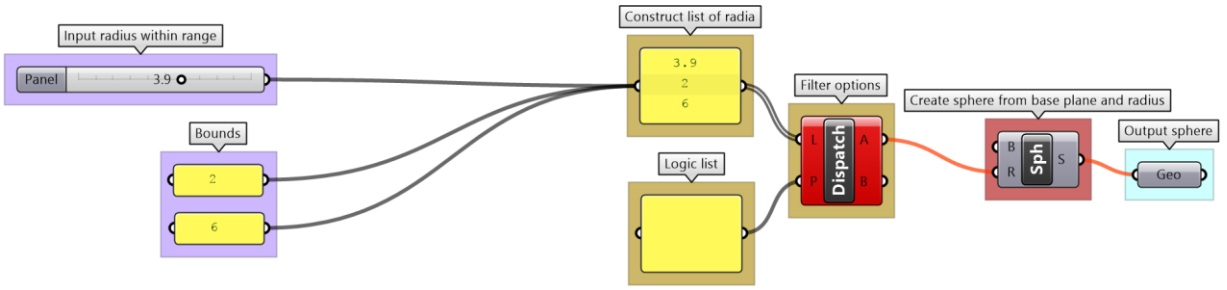
Junte todos los procesos



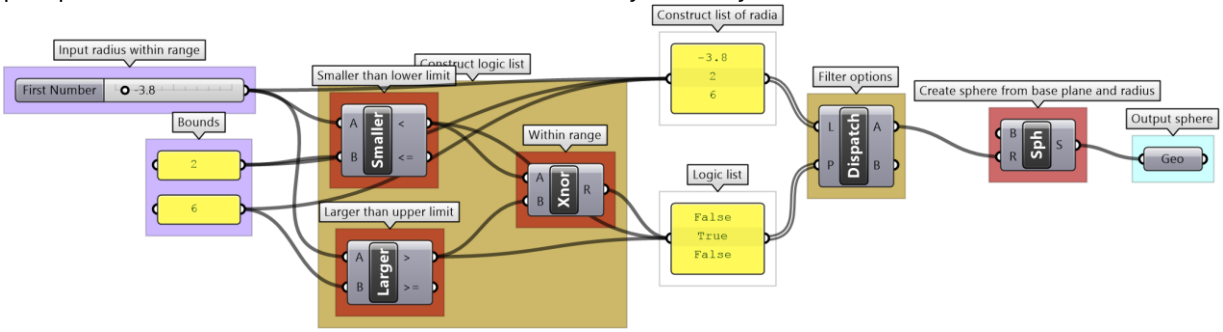
1_9_2: Tutorial de esfera con límites

Utilice el proceso de 4 pasos para dibujar una esfera con un radio entre 2 y 6. Si la entrada es menor que 2, establezca un radio de 2, y si el radio de entrada es mayor que 6, establezca un radio de 6. Utilice un control deslizante numérico para introducir el radio y establézcalo entre 0 y 10 para la prueba.
Asegúrese de que su solución esté bien organizada, codificada por colores y etiquetada correctamente.

Proceso de 4 pasos para resolver los algoritmos	
Salida La esfera como geometría	
Proceso clave: crear una esfera Mapee la lista de coordenadas de su dominio actual a un nuevo dominio 3 a 9 Utilice el componente ***ReMap***	
Entrada 1- El parámetro del radio (0 - 10) 2- Los límites del radio son 2 y 6	
Procesos intermedios 1 Cree una lógica de selección de radios y patrones. Los radios son una lista de los valores del control deslizante, mínimo y máximo. La lista de patrones se genera para seleccionar el valor correcto del radio.	

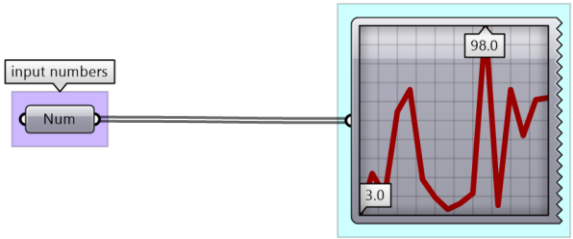
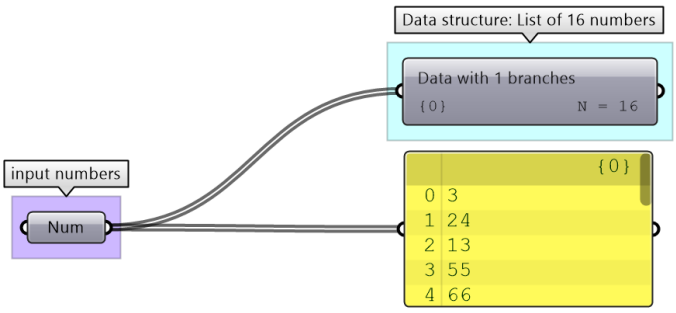


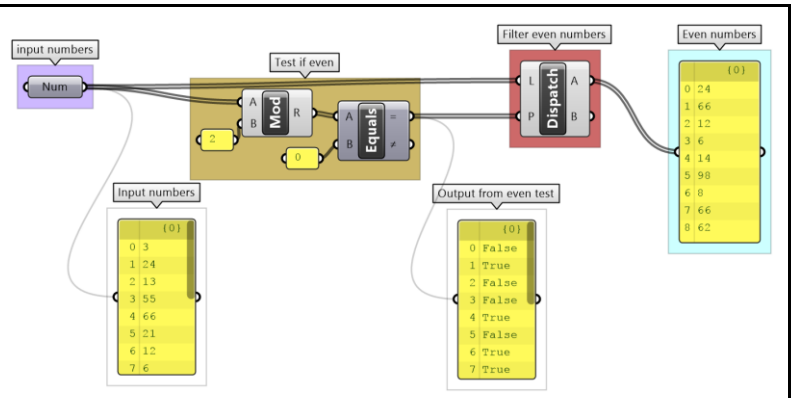
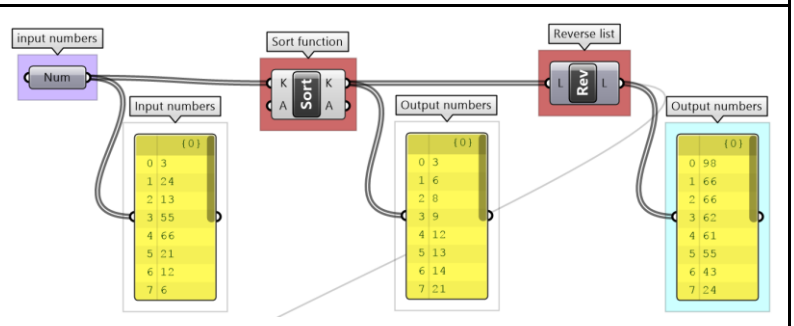
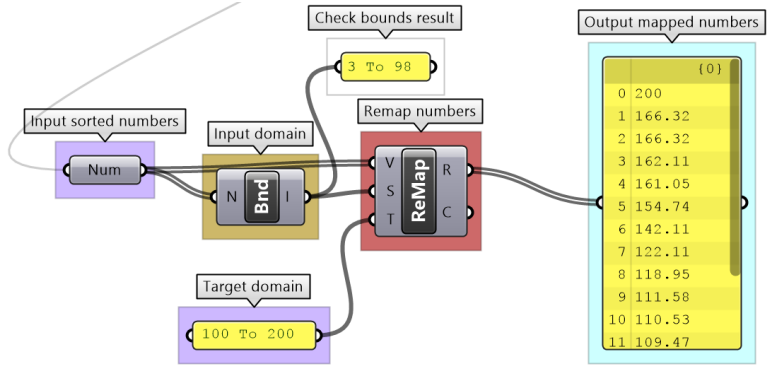
Procesos intermedios 2
La lógica de selección comprueba si el radio del control deslizante se encuentra entre los límites y lo establece para para la selección. Si es menor, seleccione el mínimo, y si es mayor, seleccione el máximo.



1_9_3: Tutorial de operaciones de datos

Dados los números insertados en el parámetro de número más abajo:
1- Analice los límites y la distribución de la entrada
2- Vea los datos y cómo están estructurados
3- Extraiga los números pares
4- Ordene los números de forma descendente
5- Reasigne los números ordenados (de 100 a 200)

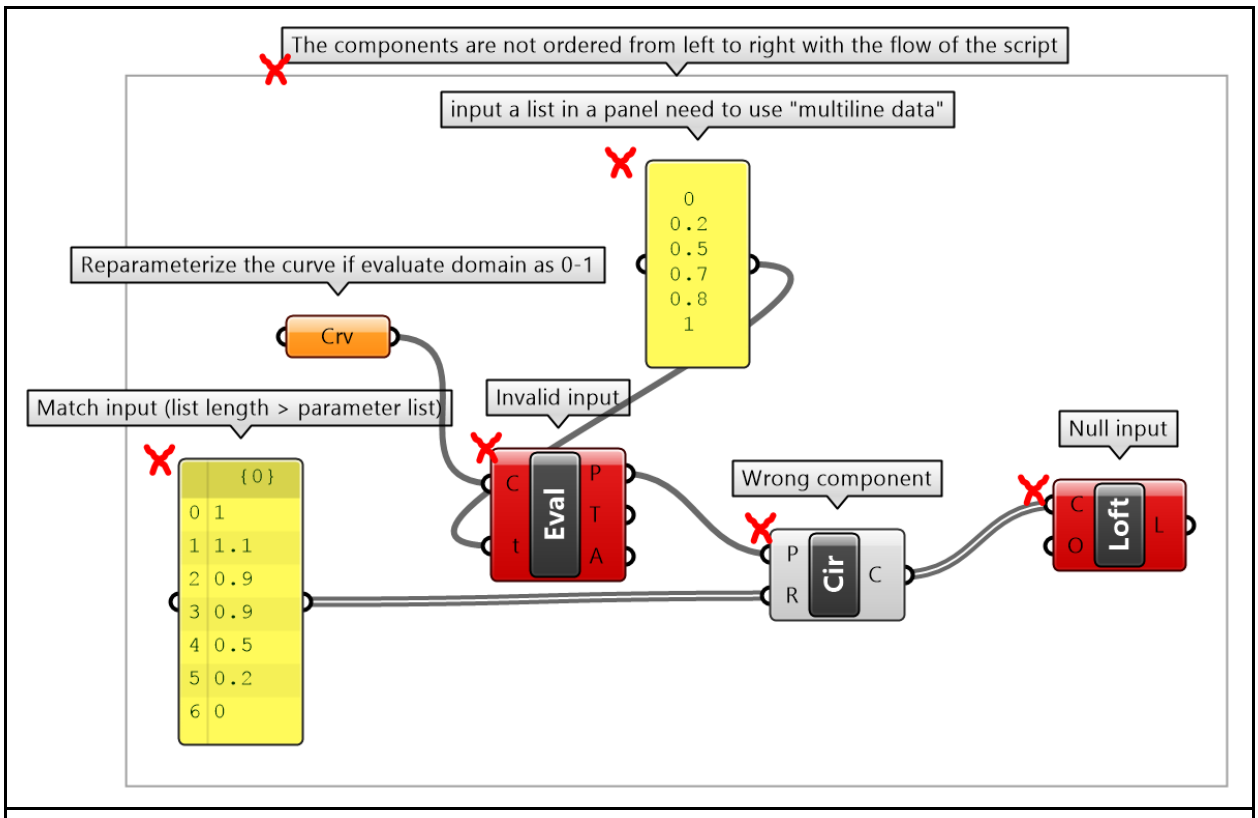
Solución	
1- Analice los límites y la distribución de la entrada Utilice el **QuickGraph** para mostrar que el conjunto de números está entre 3 y 98 y se distribuye aleatoriamente.	
2- Analice la estructura y los valores de los datos de entrada Utilice los componentes ***Panel*** y ***Parameter Viewer*** para mostrar que hay 16 elementos organizados en una lista.	

3- Extraiga los números pares Cree la lógica para probar si un número es par (divisible por 2 sin resto) y utilice ***Dispatch*** para extraer los números pares.	
4- Ordene los números de forma descendente El componente ***Sort List*** ordena los números en orden ascendente. Utilice el componente ***Reverse List*** para seguir procesando la lista y ordenarla de forma descendente.	
5- Remapear a 100-200 Compruebe el intervalo de entrada y utilice el componente Remap para escalar los datos y que estén entre 100-200.	

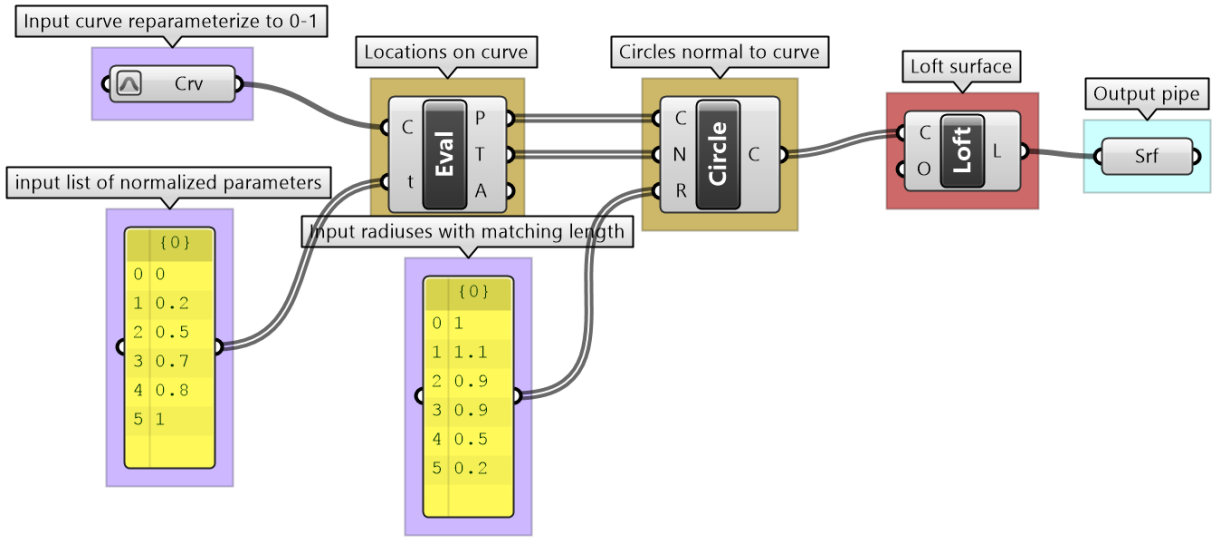
1_9_4: Tutorial para resolver dificultades

Analice lo que quiere que haga el siguiente algoritmo, identifique los errores que impiden que funcione correctamente y, a continuación, reescríbalo para corregir los errores. Organice la información para que refleje el flujo del algoritmo, las etiquetas y la codificación por colores.

Solución
Marque los errores:



Corrija los errores y reescriba la solución con etiquetas:



Capítulo 2: Introducción a las estructuras de datos

Todos los algoritmos implican procesar datos de entrada para generar un nuevo conjunto de datos como salida. Los datos se almacenan en estructuras bien definidas para facilitar su acceso y manipulación. Comprender estas estructuras es clave para que los diseños algorítmicos funcionen. Este capítulo incluye una revisión en profundidad de las estructuras de datos básicas en Grasshopper.

2_1: Resumen

Grasshopper tiene tres estructuras de datos distintas: elemento **único**, **lista** de elementos y **árbol** de elementos. Los componentes de GH se ejecutan de forma diferente en función de las estructuras de datos de entrada, por lo que es esencial conocer bien la estructura de los datos antes de utilizarlos. Existen herramientas en GH que ayudan a identificar la estructura de los datos. Son el ***Panel*** y el ***Visor de parámetros***.

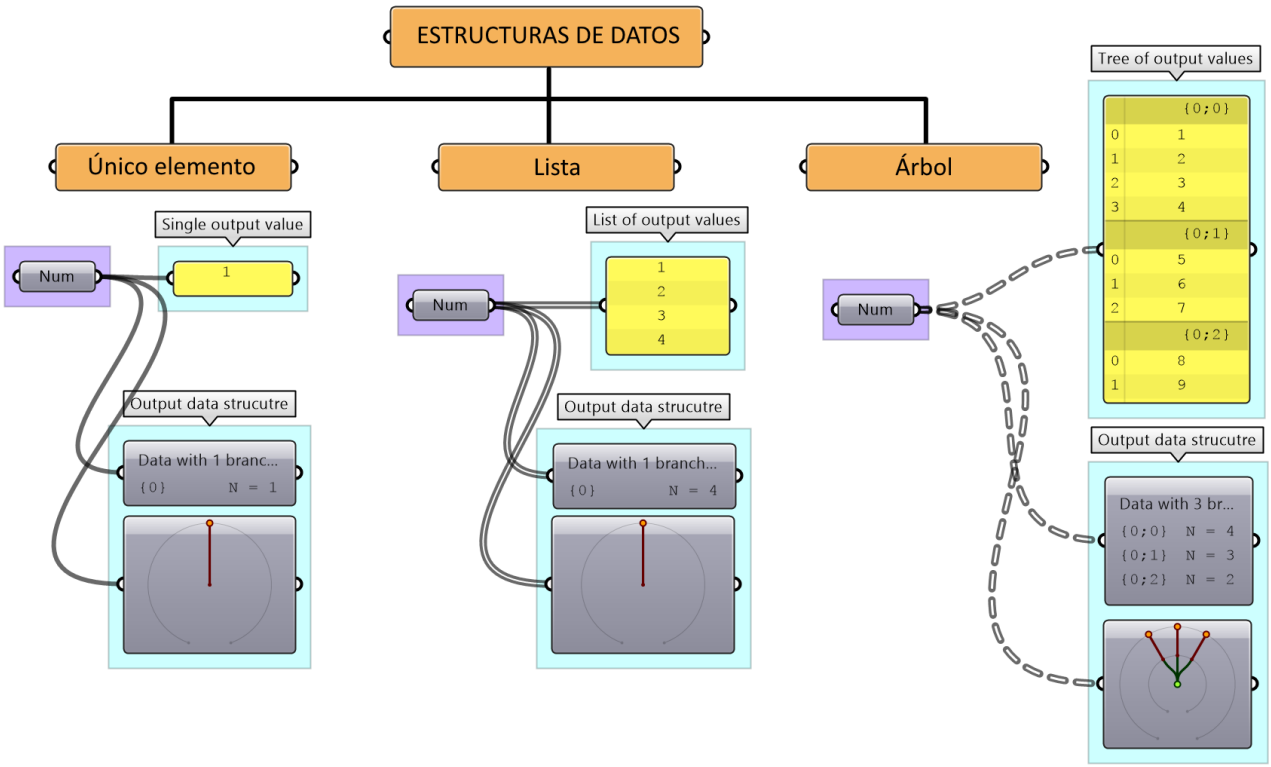


Figura (34): Estructuras de datos en Grasshopper

Los procesos en GH se ejecutan de forma diferente en función de la estructura de datos. Por ejemplo, el componente ***Mass Addition*** suma todos los números de una lista y genera un número único, pero al operar sobre un árbol, produce una lista de números que representan la suma de cada rama.

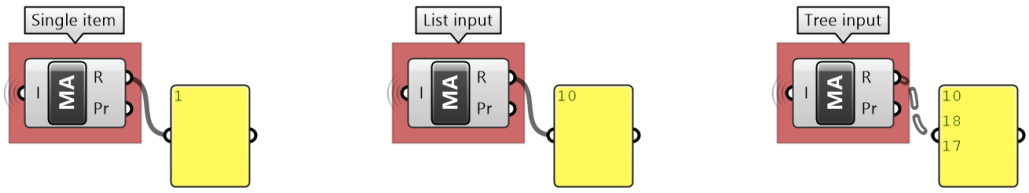
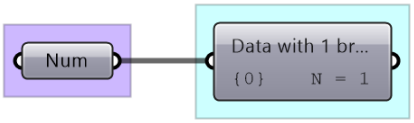
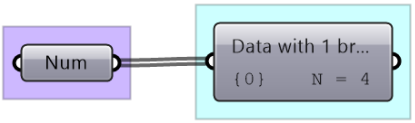
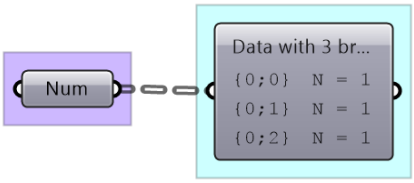


Figura (35): Los componentes se ejecutan de forma diferente en función de las estructuras de datos. Resultado de la suma de números de la figura (34)

Los cables que conectan los datos con los componentes en GH ofrecen una referencia visual adicional a la estructura de datos. El cable de un elemento único es una línea simple, mientras que el cable que conecta una lista se dibuja mediante una línea doble. La salida de cable de una estructura de datos en forma de árbol es una línea doble discontinua. Este sistema es muy útil para identificar rápidamente la estructura de los datos.

Visualización de la estructura de datos	**Ejemplo**
Elemento: rama única con un elemento único **Tipo de cable**: una línea	
Lista: rama única con varios elementos **Tipo de cable**: doble línea	
Árbol: varias ramas con cualquier número de elementos por rama **Tipo de cable**: línea doble discontinua	

2_2: Generación de listas

Hay muchas formas de generar listas de datos en GH. Hasta ahora hemos visto cómo insertar directamente una lista de valores dentro de un parámetro o un panel (con datos multilínea). También hay componentes especiales para generar listas. Por ejemplo, para generar una lista de números, hay tres componentes clave: ***Range*** (Rango), ***Series*** (Serie) y ***Random*** (Aleatorio). El componente ***Range*** crea un rango de números igualmente espaciados entre un valor mínimo y un valor máximo (llamado dominio) y un número de pasos (el número de valores de la lista resultante es igual al número de pasos más uno).

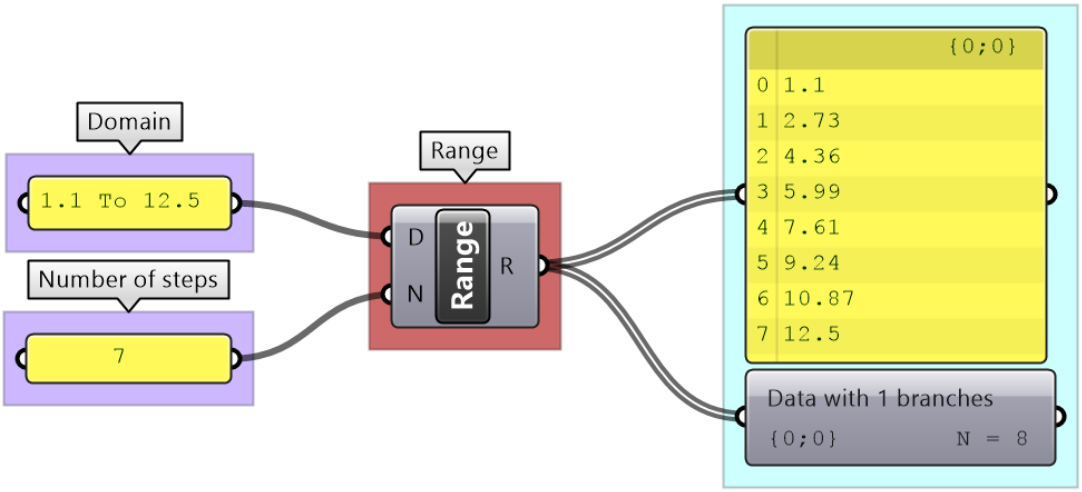


Figura (36): Generación de una lista de 8 números mediante el componente ***Range*** en Grasshopper

El componente ***Series*** también crea una lista de números igualmente espaciados, pero aquí el usuario establece el número inicial, el tamaño del paso y el número de elementos.

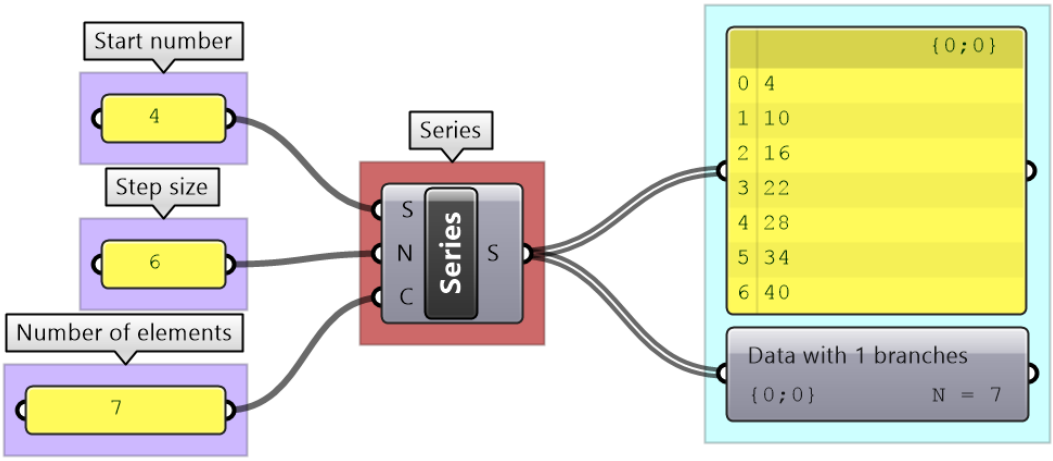


Figura (37): Generación de una lista de 7 números mediante el componente ***Series*** en Grasshopper

El componente ***Random*** se utiliza para crear números aleatorios utilizando un dominio y un número de elementos. Si se utiliza la misma semilla, siempre se obtiene el mismo conjunto de números aleatorios.

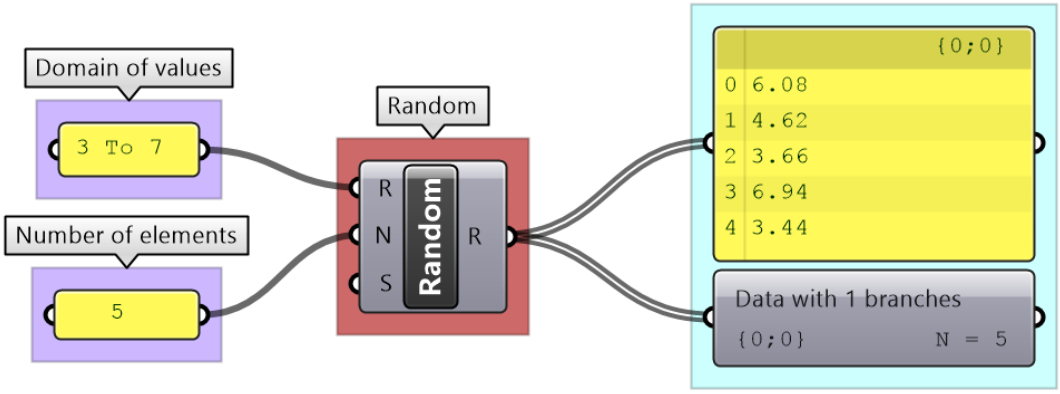


Figura (38): Generación de una lista de números mediante el componente ***Random*** en Grasshopper

Las listas pueden ser la salida de algunos componentes como ***Divide*** (la salida incluye listas de puntos, tangentes y parámetros). Utilice el componente ***Panel*** para previsualizar los valores en una lista y ***Parameter Viewer*** para examinar las estructuras de datos.

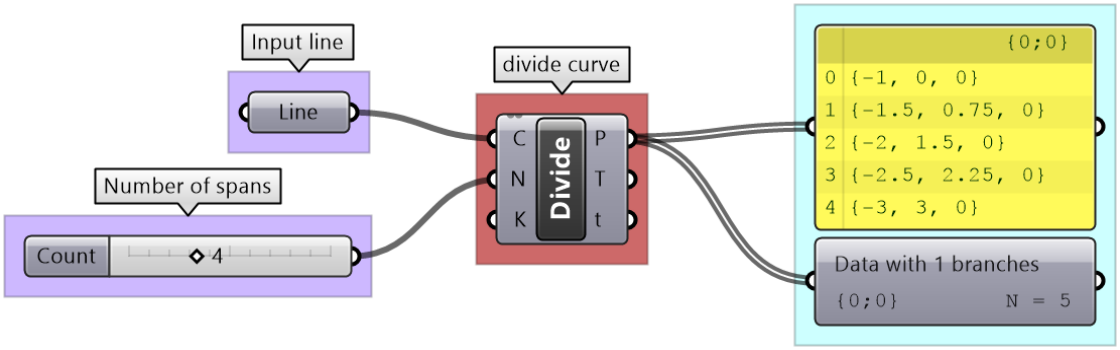
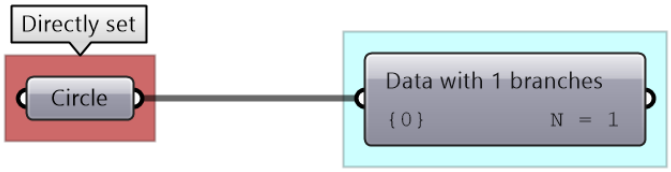
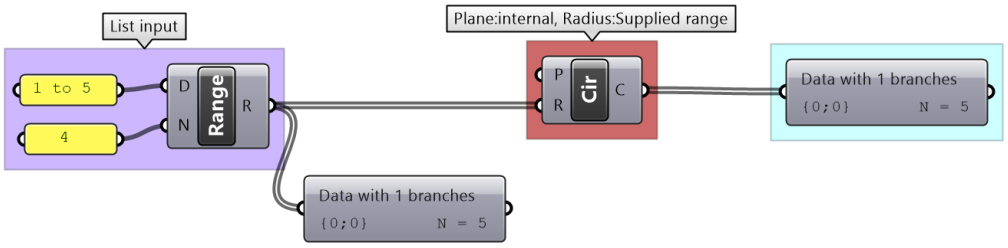
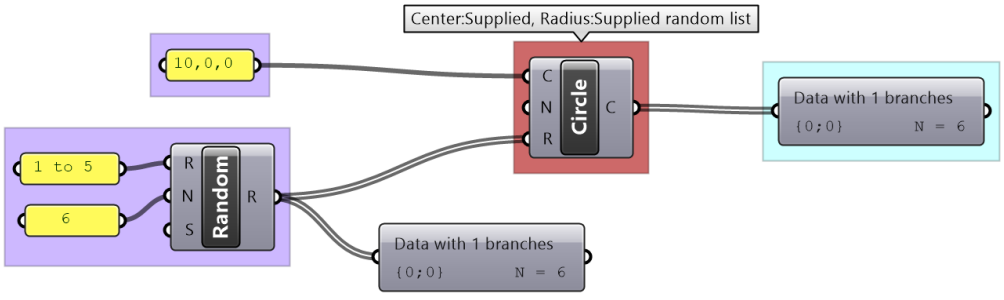
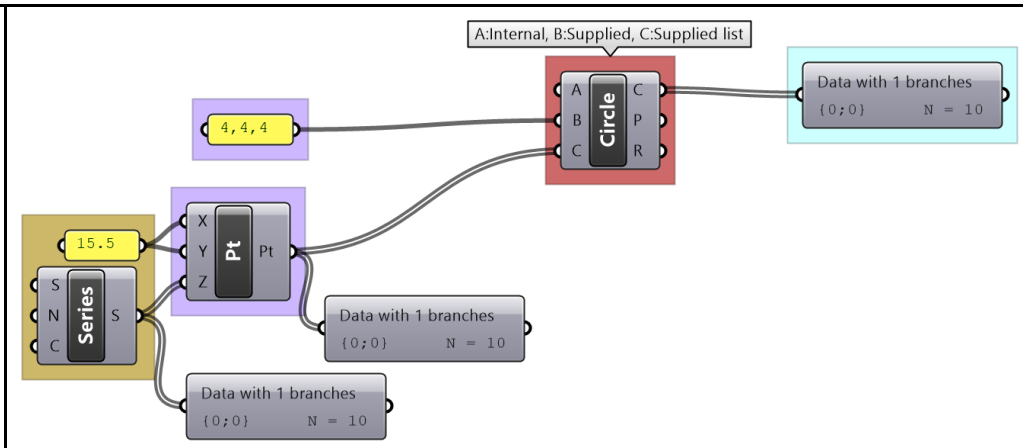


Figura (39): ***Divide*** toma una entrada única (curva) y genera listas de salida

2_2_1 Tutorial de generación de listas

Explore 4 formas diferentes de crear círculos. Utilice diferentes fuentes y estructuras de datos.

Descripción	**Solución de Grasshopper**
Defina directamente un círculo en un parámetro.	
Establezca la entrada del plano en el plano XY predeterminado (interno). Proporcione una lista de radios utilizando el componente ***Range***.	
Proporcione un valor para el centro. La normal está definida como predeterminada (interna). Lista de radios utilizando el componente ***Random***.	

Círculo a partir de tres puntos: A: defínalo internamente a un valor B: proporcione un valor C: proporcione una lista de valores utilizando el componente ***Series*** para definir una lista de coordenadas Z	

2_3: Operaciones con listas

GH ofrece una amplia gama de componentes para operaciones con listas y gestión de listas. Vamos a evaluar algunos de los más utilizados.
Puede comprobar la longitud de una lista mediante el componente ***List Length*** y acceder a los elementos en índices específicos mediante el componente ***List Item***.

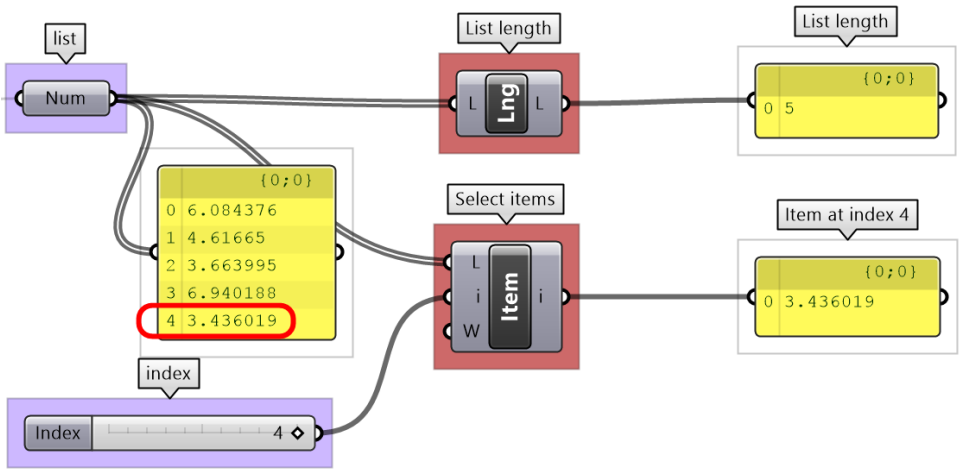


Figura (40): Ejemplos de operaciones con listas en Grasshopper

Las listas pueden invertirse con el componente ***Reverse List*** y ordenarse con el componente ***Sort List***.

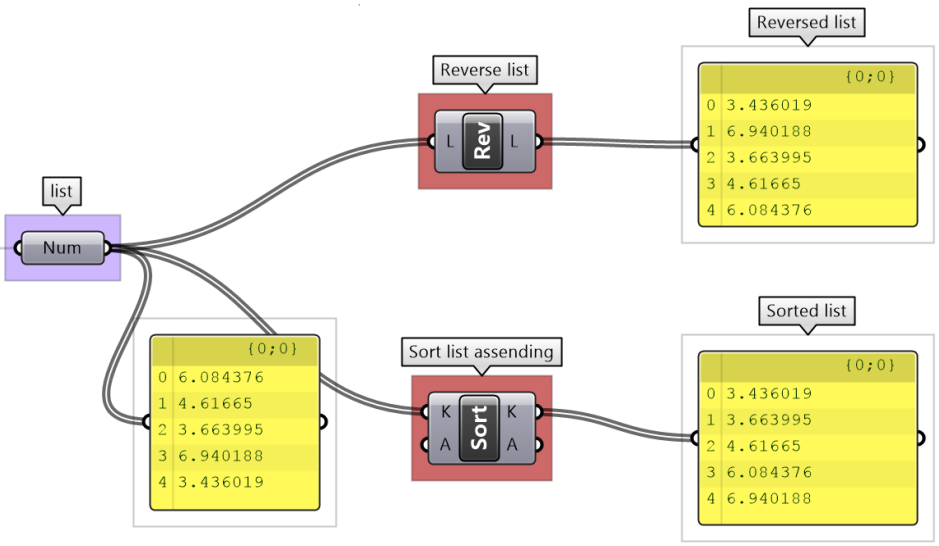


Figura (41): Las listas pueden ordenarse o invertirse utilizando los componentes correspondientes en Grasshopper

Los componentes ***Cull Patterns*** y ***Dispatch*** permiten seleccionar un subconjunto de la lista o dividir la lista basándose en un patrón. Estos componentes se utilizan a menudo para controlar el flujo de datos y seleccionar un subconjunto de datos.

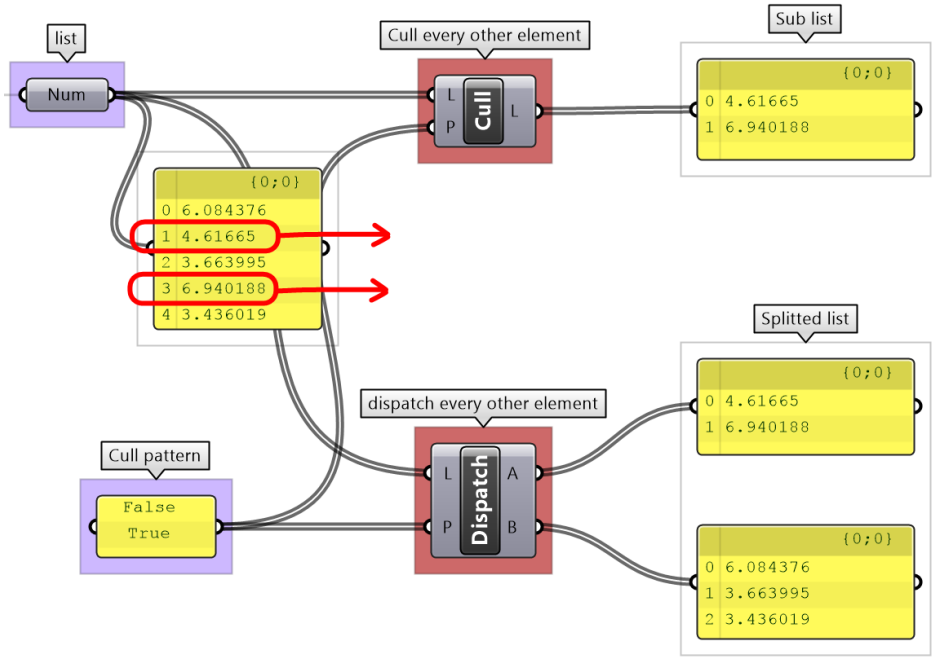


Figura (42): Separe parte de una lista utilizando los componentes ***Cull Pattern*** y ***Dispatch***

El componente ***Shift List*** permite mover los elementos de una lista con un número determinado de pasos. Este componente permite alinear varias listas para que coincidan en un orden determinado.

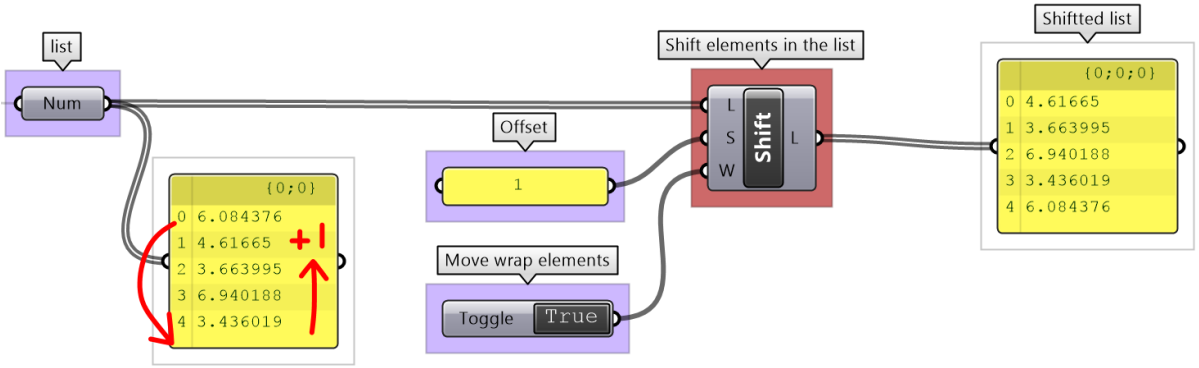


Figura (43): Operación con el componente ***Shift*** en Grasshopper

El componente ***Subset*** es otra herramienta que sirve para seleccionar parte de una lista basada en un intervalo de índices.

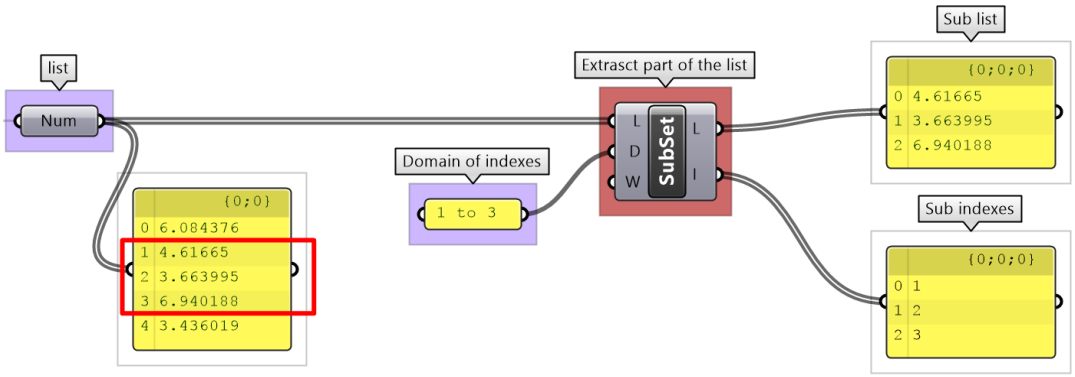


Figura (44): Ejemplo de selección de un subconjunto de la lista mediante un intervalo de índices

2_3_1 Tutorial de operaciones con listas

Utiliza las dos listas de puntos dadas para generar las siguientes imágenes.

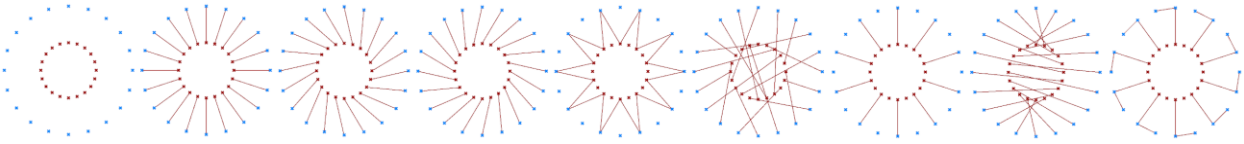
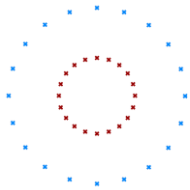
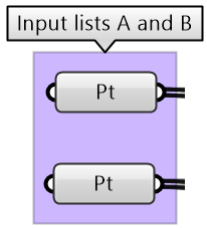
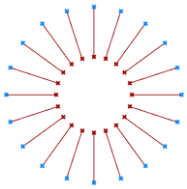
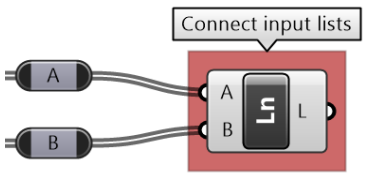
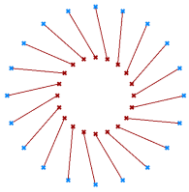
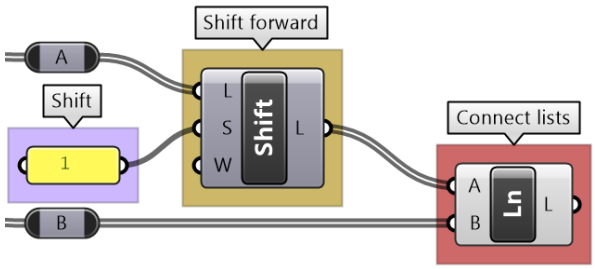
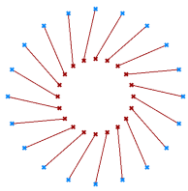
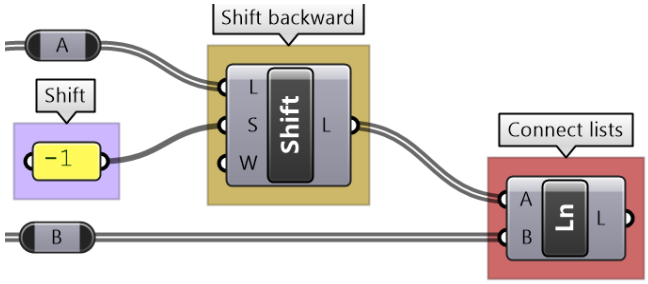


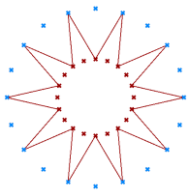
Imagen de salida	Solución de Grasshopper
	
	

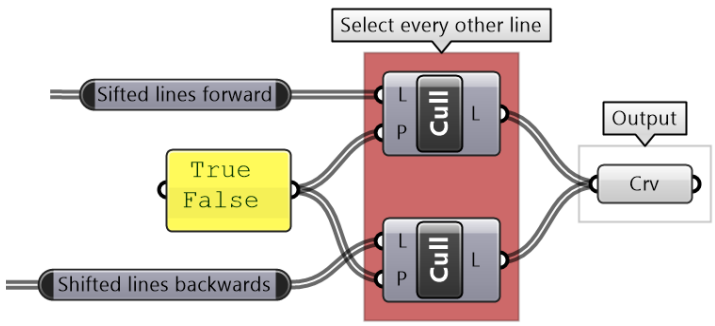


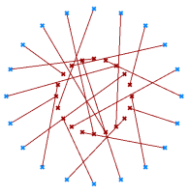

Shift forward
A
Shift
1
B
L
S
W
Shift
L
Connect lists
A
B
Ln
L

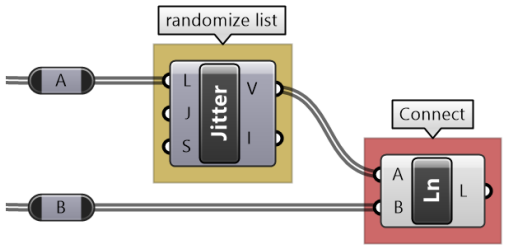


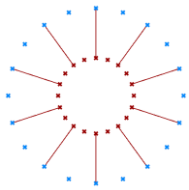

Shift backward
A
Shift
-1
B
L
S
W
Shift
L
Connect lists
A
B
Ln
L

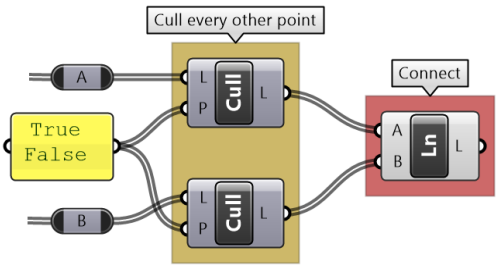


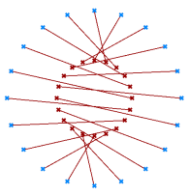

Select every other line
Sifted lines forward
True
False
Shifted lines backwards
L
P
Cull
L
Output
Crv

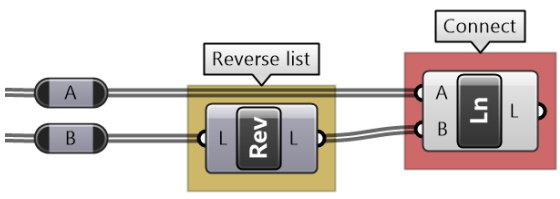


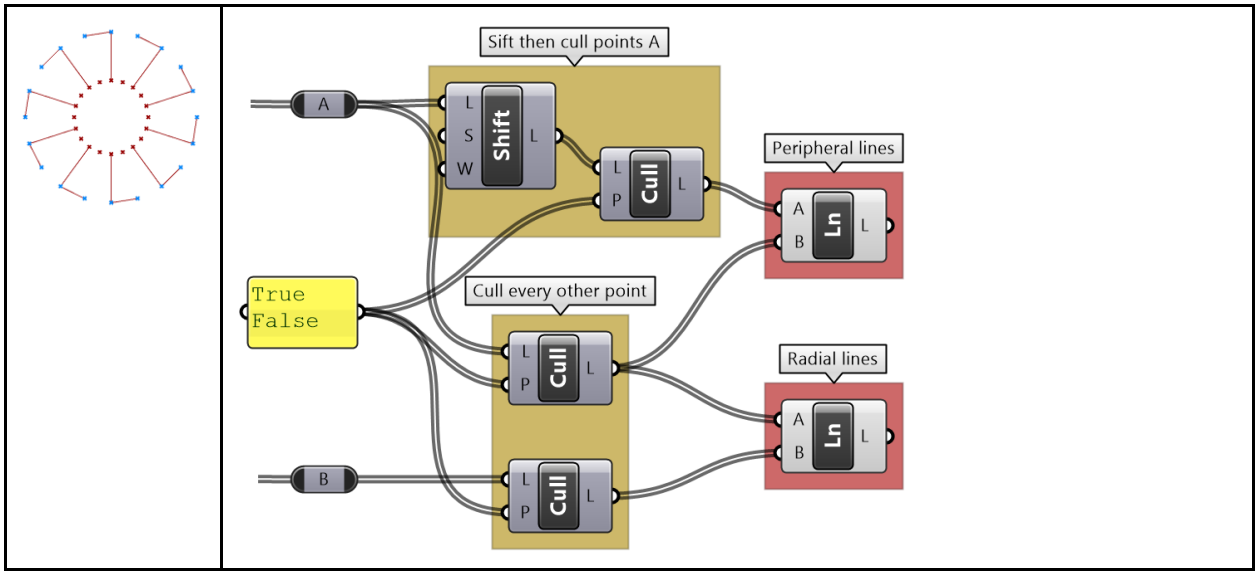

randomize list
A
B
L
J
S
Jitter
V
I
Connect
A
B
Ln
L




Cull every other point
A
True
False
B
L
P
Cull
L
Connect
A
B
Ln
L




Connect
Reverse list
A
B
L
Rev
L
A
B
Ln
L



2_4: Correspondencia de listas

Cuando la entrada es un elemento único o tiene un número igual de elementos en una lista simple, es fácil que los datos se correspondan. La correspondencia se basa en índices coincidentes. Utilicemos el componente ***Addition*** para examinar la correspondencia de listas en GH.

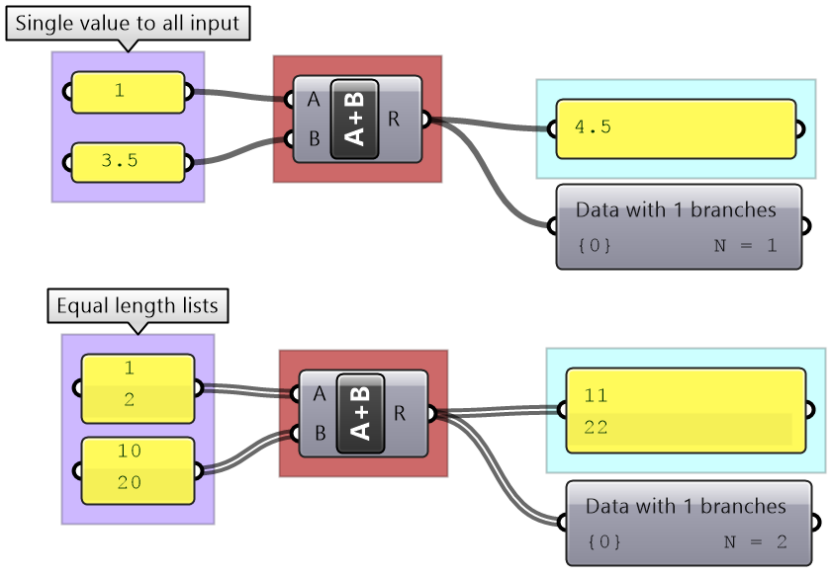


Figura (45): La correspondencia de listas de longitudes iguales se basa en hacer corresponder los índices coincidentes

Hay veces que la entrada tiene listas de longitud variable. En este caso, GH reutiliza el último elemento de la lista más corta y lo hace coincidir con los siguientes elementos de la lista más larga.

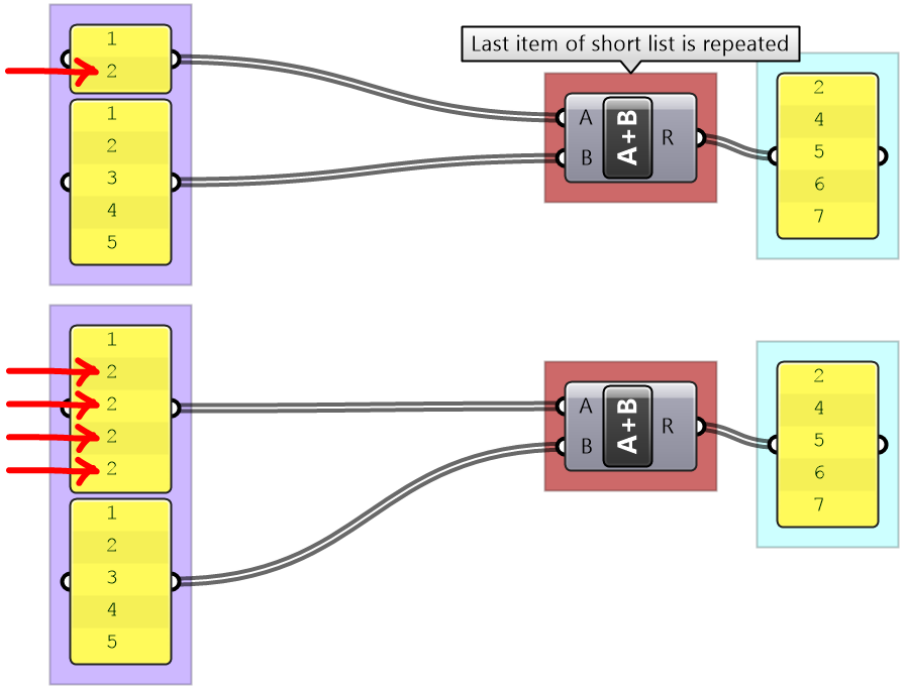


Figura (46): La correspondencia de listas por defecto en Grasshopper reutiliza el último elemento de la lista más corta

Grasshopper ofrece formas alternativas de correspondencia de datos que el usuario puede utilizar: referencias ***largas***, ***cortas*** y ***cruzadas***. La correspondencia ***larga*** es igual que la predeterminada. Es decir, el último elemento de la lista más corta se repite para crear una longitud coincidente.

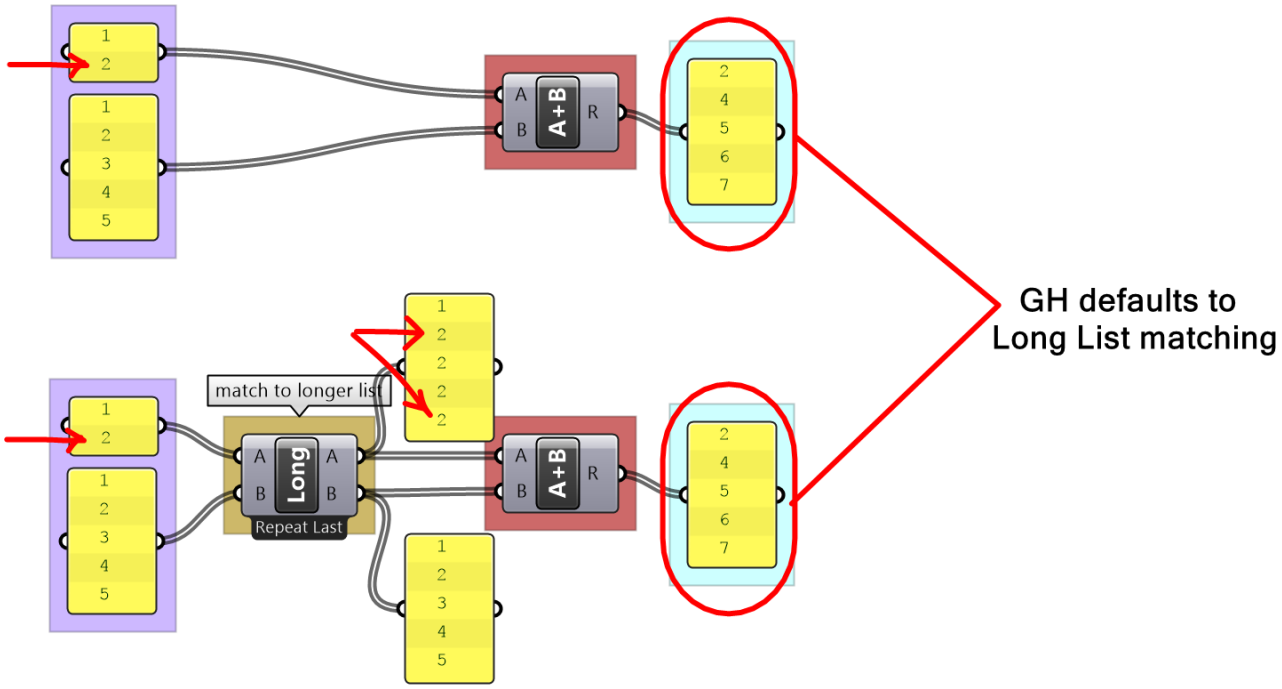


Figura (47): La correspondencia de lista larga es el modo de correspondencia por defecto en Grasshopper

La correspondencia de lista ***corta*** trunca la lista larga para que coincida con la longitud de la lista corta. Todos los elementos adicionales se ignoran y la lista resultante tiene una longitud que coincide con la lista más corta.

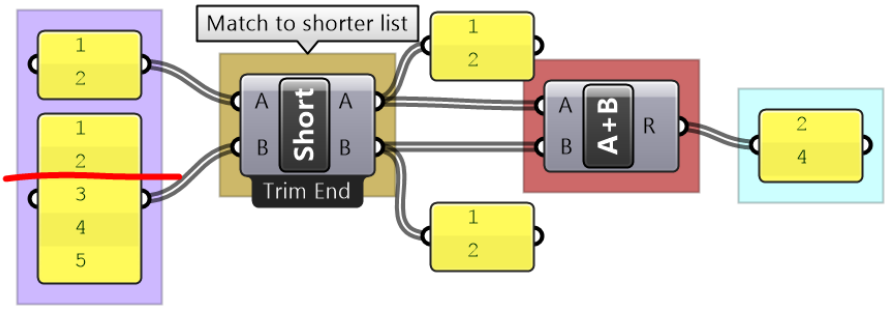


Figura (48): La correspondencia de listas cortas omite los valores adicionales en las listas más largas

El componente ***Cross Reference*** hace coincidir la primera lista con cada uno de los elementos de la segunda lista. La lista resultante tiene una longitud igual al producto multiplicador de la longitud de las listas de entrada. Las referencias cruzadas son útiles cuando se quiere producir todas las combinaciones posibles de datos de entrada. El orden de entrada afecta al orden del resultado, como se muestra en la figura (49).

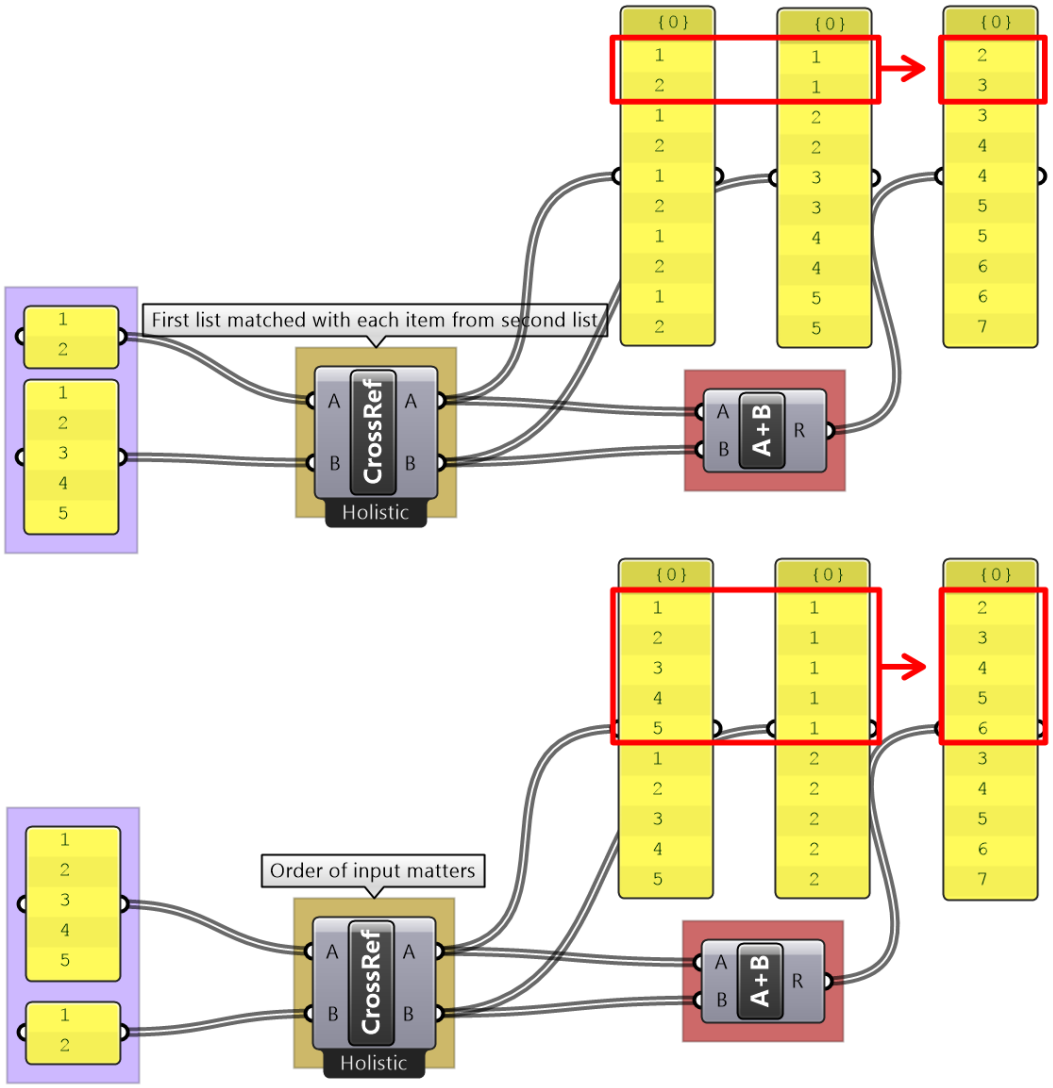


Figura (49): La correspondencia de referencias cruzadas crea listas más largas para tener en cuenta todas las combinaciones posibles

Si ninguno de los métodos de correspondencia produce el resultado deseado, puede ajustar explícitamente las listas para que coincidan en longitud según sus necesidades. Por ejemplo, si quiere repetir la lista más corta hasta que coincida con la longitud de la lista más larga, tendrá que crear la lógica para lograrlo como se muestra en el siguiente ejemplo.

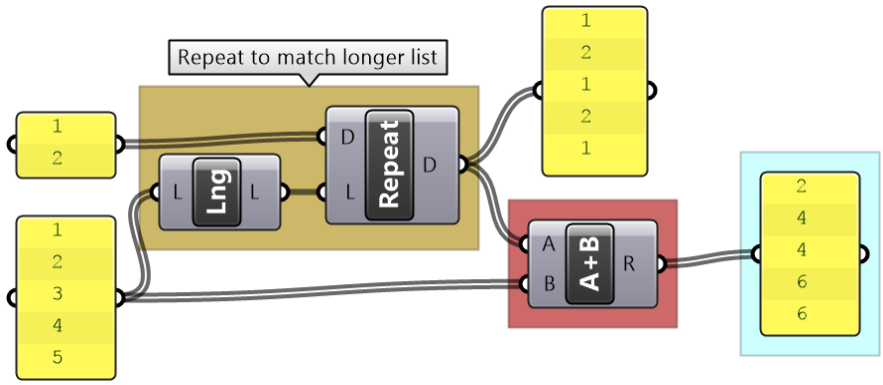
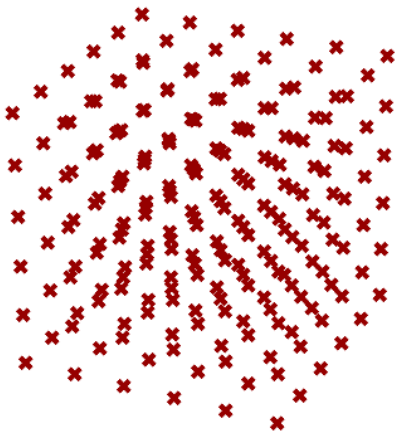
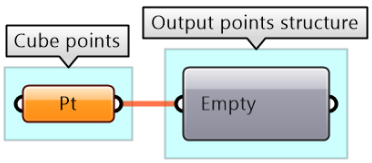
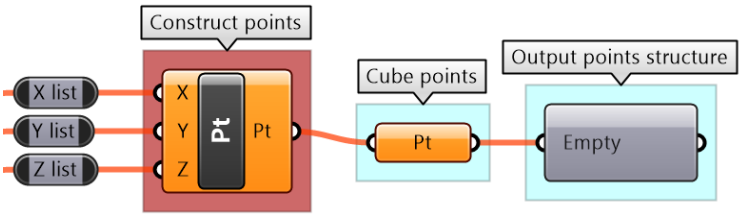


Figura (50): Es necesario crear un script personalizado para generar una correspondencia personalizada

2_4_1 Tutorial de correspondencia de listas

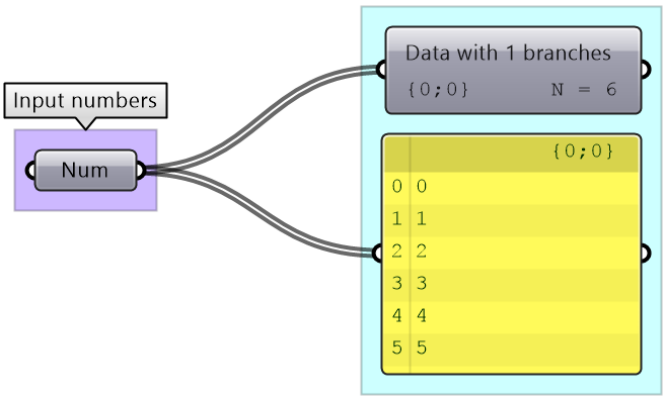
Utilice la lista de entrada de 6 números para crear los puntos de la imagen.



Solución	
Salida: Una lista de 6x6x6 = 216 puntos creada a partir de una lista de coordenadas X, Y, Z.	
Proceso clave: Utilice el componente ***Construct Point*** para generar la lista de puntos.	

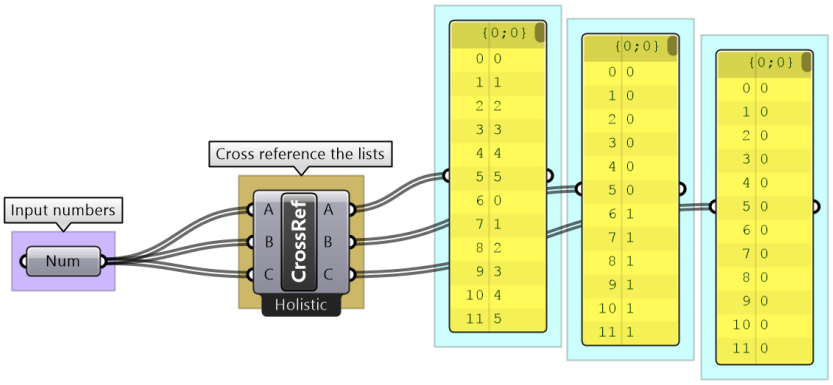
Entrada:
Examine la entrada utilizando los componentes ***Parameter Viewer*** y ***Panel***.

La lista dada tiene 6 puntos que representan cada coordenada a lo largo de cada eje.

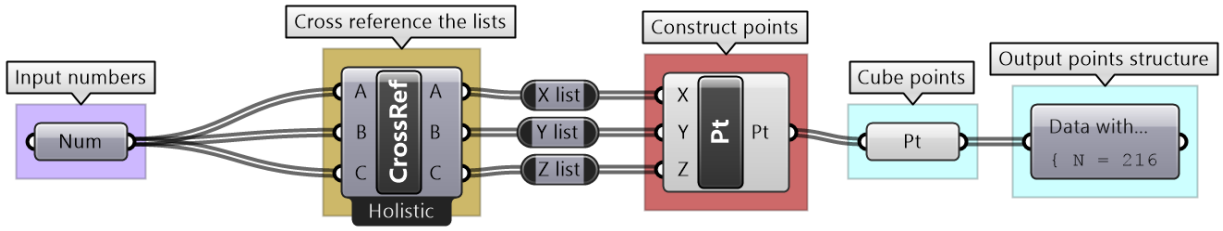


Proceso intermedio:
Hay que encontrar todas las permutaciones posibles de las coordenadas para crear el cubo de 216 puntos a lo largo de los 3 ejes.

Utilice la correspondencia de ***referencias cruzadas*** para generar listas de coordenadas que tengan todas las combinaciones posibles.



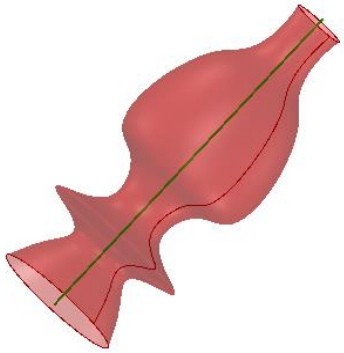
Junte todos los procesos



2_5: Tutoriales de estructuras de datos

2_5_1: Tutorial de un tubo de grosor variable

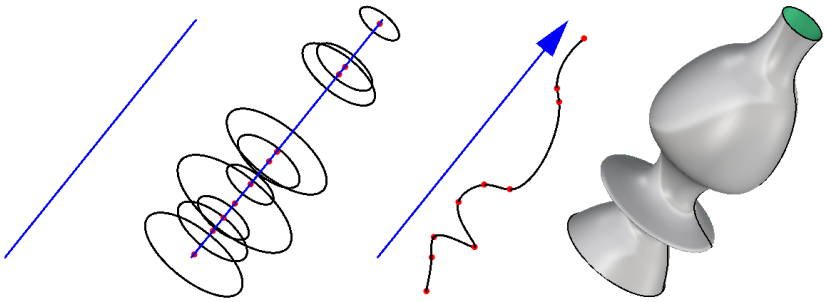
Cree una superficie similar a la de la imagen con un grosor que cambie en 10 lugares al azar a lo largo de la curva. Las variaciones de grosor son aleatorias entre 1 y 3.



Análisis del algoritmo

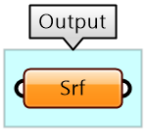
Para calcular un algoritmo, es útil pensar en términos de modelado 3D. Hay dos maneras de generar esta superficie:
1- Cree **círculos** a lo largo de la línea en lugares al azar con radios aleatorios y luego ejecute el comando **Transición** en el resultado.
2- Calcule la **curva del perfil** y **revolucione** a lo largo de la línea

El primer proceso va así:
1- **Divida** la línea en posiciones aleatorias
2- Oriente a los **planos** en las posiciones (línea normal a los planos)
3- Cree los **círculos** (o **puntos** para la curva de perfil)
4- Seleccione los círculos (en orden) para usar el componente **Loft** (o **Interpolate Curve** y luego **Revolve**)

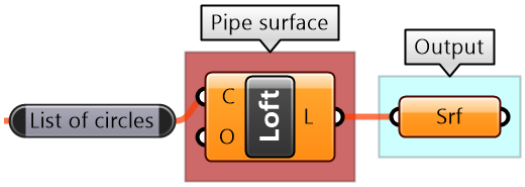


Pasos de la solución

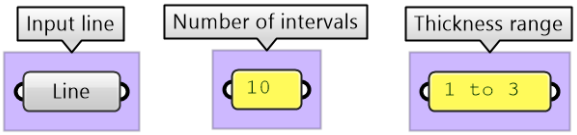
Salida:
La superficie



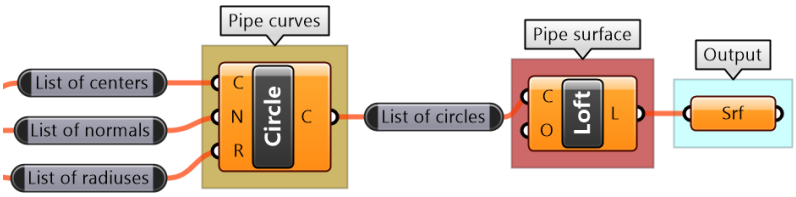
Proceso clave:
Utilice el componente ***Loft*** para generar la superficie



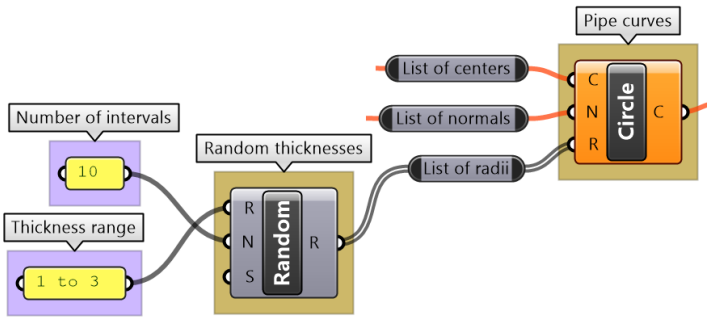
Entrada:
Línea,
número de intervalos e intervalo de grosor



Proceso intermedio 1:
El ***Loft*** se crea a partir de una lista de círculos. Utilice el componente ***Circle*** que toma listas de centros, normales y radios.
Podemos usar las opciones predeterminadas de ***Loft***.

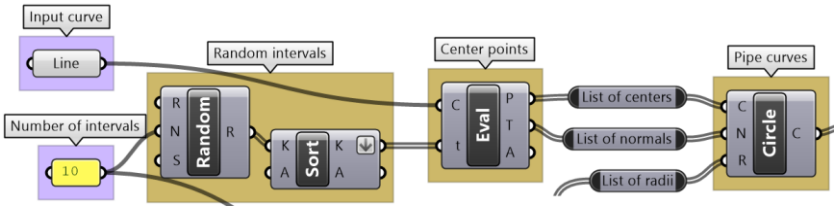


Proceso intermedio 2:
La lista de radios se crea aleatoriamente. Utilice el componente ***Random*** y el intervalo de grosor de entrada.

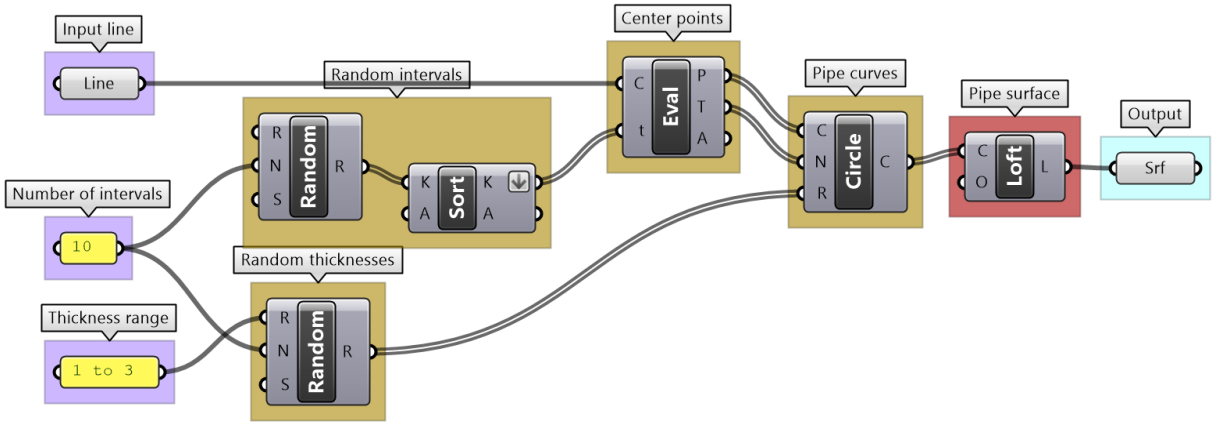


Proceso intermedio 3:
Calcule la línea a intervalos aleatorios. Utilice el componente **Evaluate Curve** para extraer los puntos y las normales, y utilice el ***Random*** para generar los parámetros a lo largo de la curva.

Problema: los parámetros aleatorios no están ordenados, por lo que producen puntos desordenados. Utilice el componente ***Sort List*** para ordenar los parámetros antes de introducirlos en ***Evaluate Curve***.



Junte todos los procesos

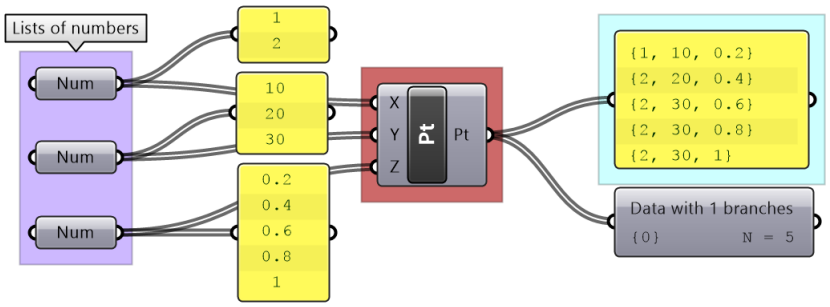


2_5_2: Tutorial de correspondencia personalizada

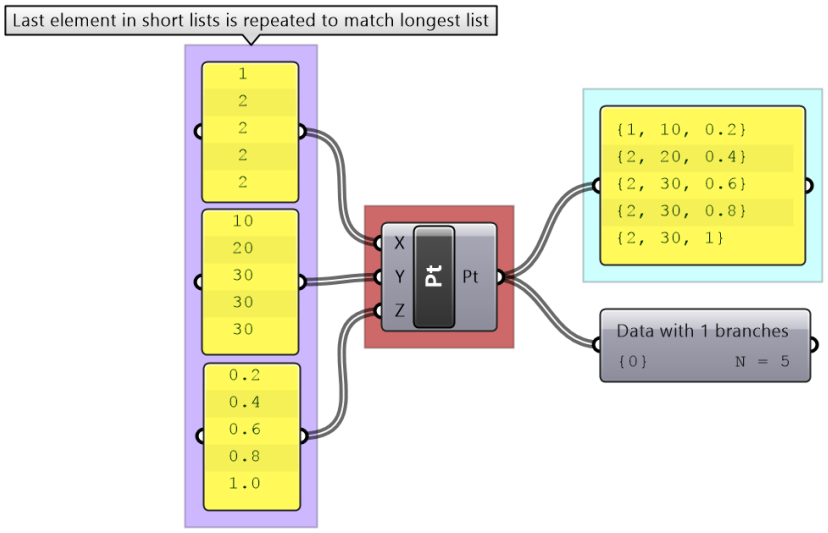
Explique la correspondencia de lista de GH predeterminada en el siguiente ejemplo. Compare el resultado con la correspondencia de "lista más corta" y, a continuación, intente crear una correspondencia personalizada que repita el patrón de las listas más cortas. Por ejemplo, [1,2] se convierte en [1,2,1,2,...] hasta que coincida con la longitud de la lista más larga.

Solución

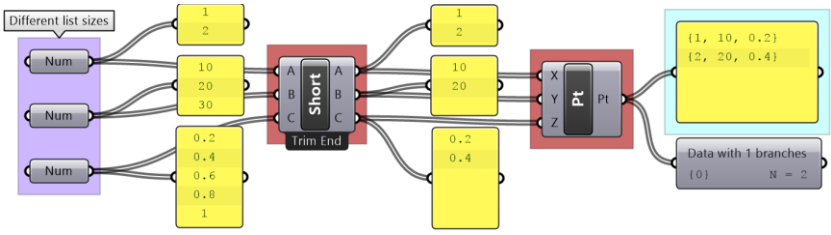
Cree la correspondencia de GH predeterminada:
Para probar la correspondencia, rellene las listas como coordenadas en un componente ***Construct Point*** y observe el resultado.



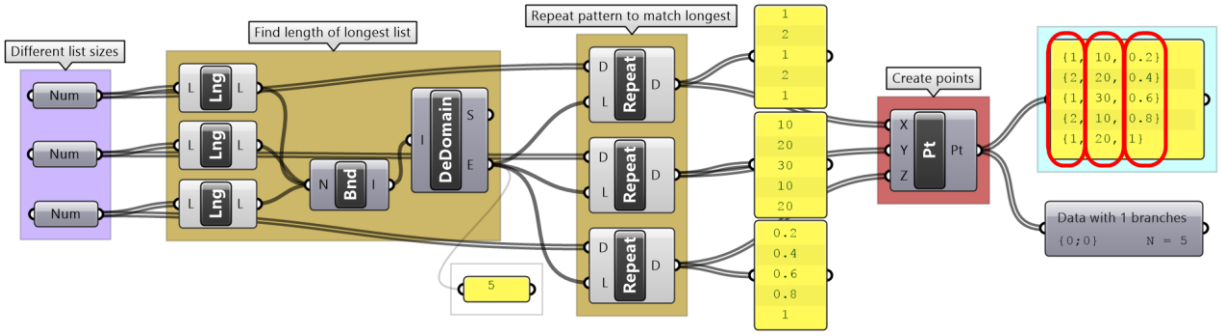
Análisis de la correspondencia de GH predeterminada:
El último elemento de las listas más cortas se repite hasta que todas las listas tengan la misma longitud; entonces los elementos se corresponden por índices.



Correspondencia de lista más corta:
Omita los valores adicionales en las listas más largas para que la longitud de todas las listas sea igual a la de la lista más corta.

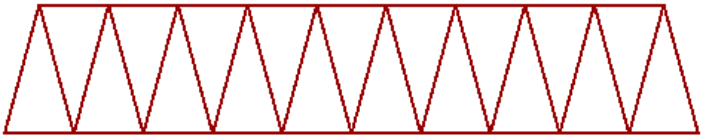
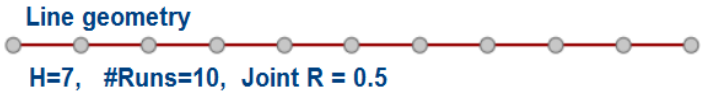
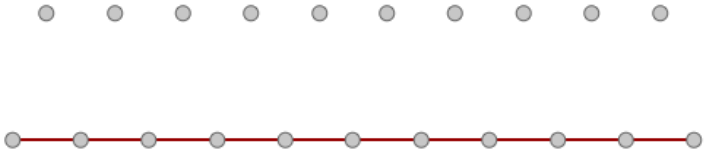
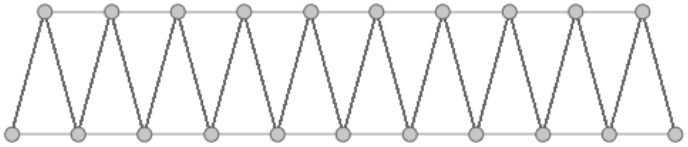
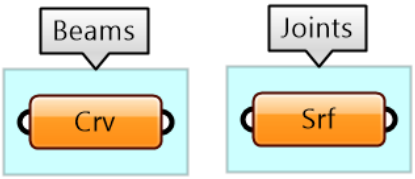


Correspondencia personalizada:
Utilice el componente ***Repeat*** para repetir los elementos hasta que coincidan con la longitud de la lista más larga.



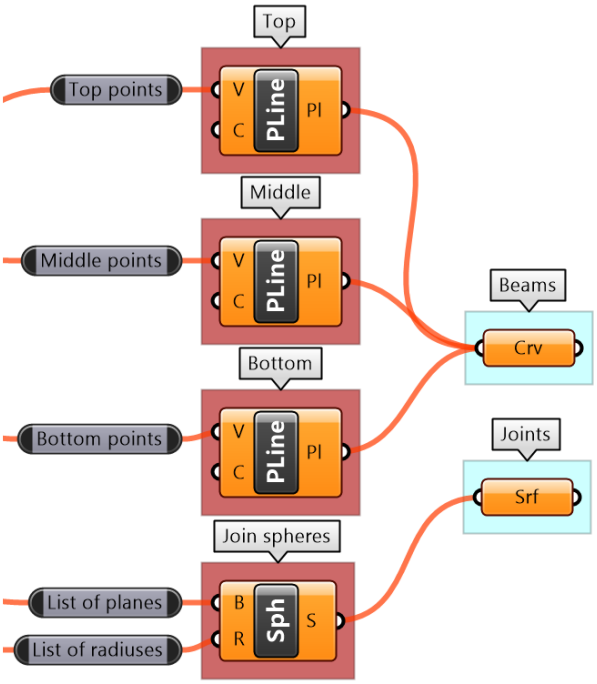
2_5_3: Tutorial de una celosía sencilla

Crea una celosía sencilla como la de la imagen. Utilice la línea de base, la altura, el número de instancias y el radio de junta.

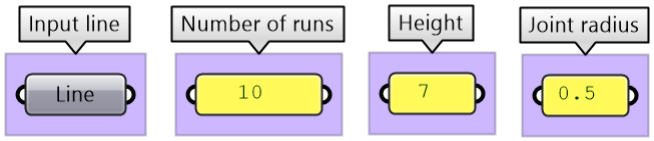
Análisis del algoritmo	
Identifique el resultado de celosía que desea	
Defina la entrada L= geometría de la línea en el plano XY A = altura I = número de instancias J = radio de junta	
Divida la curva por 2*R.	
Mueva cualquier otro punto en la dirección Z por altura.	
Cree 3 conjuntos de puntos ordenados para las vigas inferiores, las vigas superiores y las vigas centrales, y luego conecte cada uno de los 3 conjuntos con una polilínea.	
Implementación del algoritmo en Grasshopper	
Salida: Hay 2 salidas, las vigas como curvas (polilíneas) y las juntas como esferas (superficies).	

Procesos clave:
Hay que crear las polilíneas para las vigas superiores, centrales e inferiores. Utilice el componente ***Polyline*** con el correspondiente conjunto de puntos para cada uno.

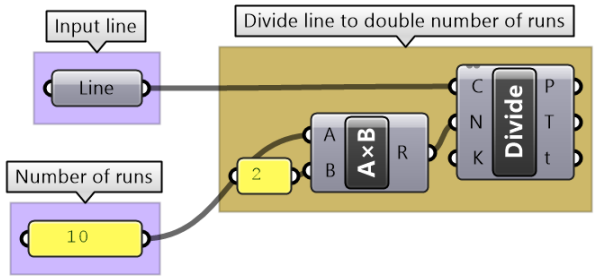
Utilice el componente ***Sphere*** para crear las juntas. Utilice los puntos medios y el radio de junta como entrada.



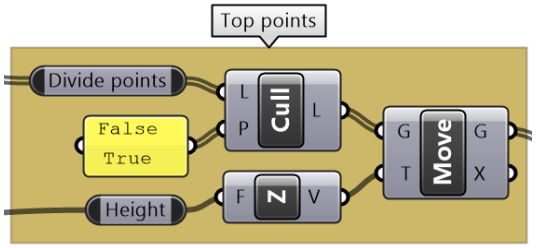
Dada la entrada:
Cuatro entradas: línea, número de instancias, altura y radio de junta.



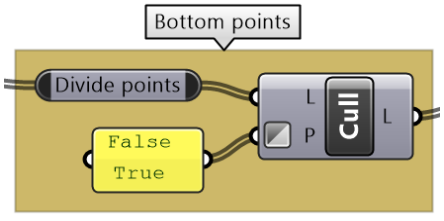
Proceso intermedio 1
Divida la curva con el doble del número de instancias. Utilice el componente ***Divide Curve*** y ***Multiply*** para multiplicar el número de instancias.

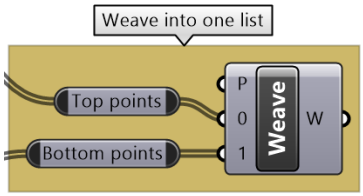
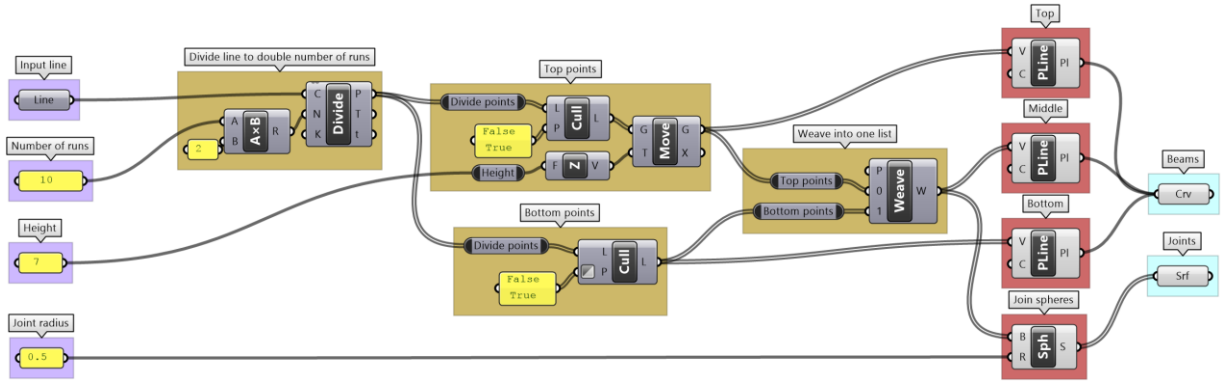


Proceso intermedio 2
Para crear puntos superiores, seleccione cualquier otro punto de la lista de los puntos de división y muévalo verticalmente la cantidad de la altura.
Utilice el componente ***Cull Pattern*** para seleccionar los puntos y ***Move*** para desplazarse verticalmente.



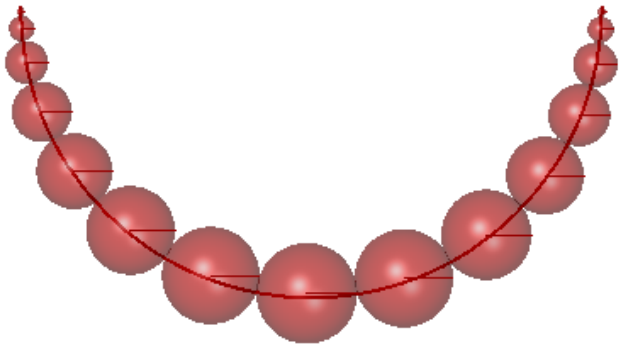
Proceso intermedio 3
Para crear los puntos inferiores, seleccione uno de cada dos puntos, en el patrón de inversión utilizado para seleccionar los puntos superiores.
Utilice el componente ***Cull***

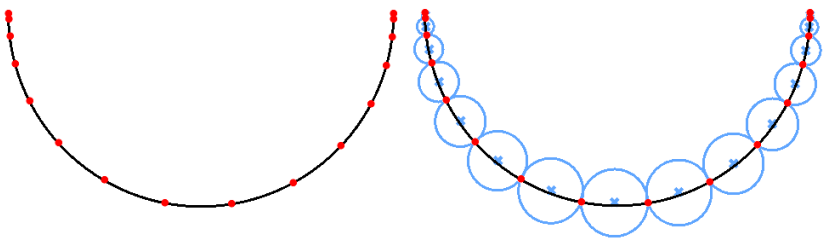


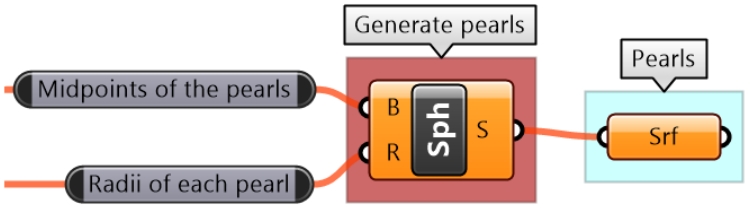
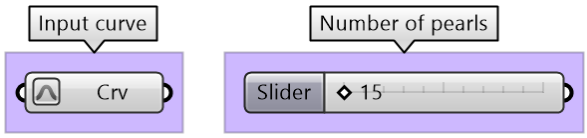
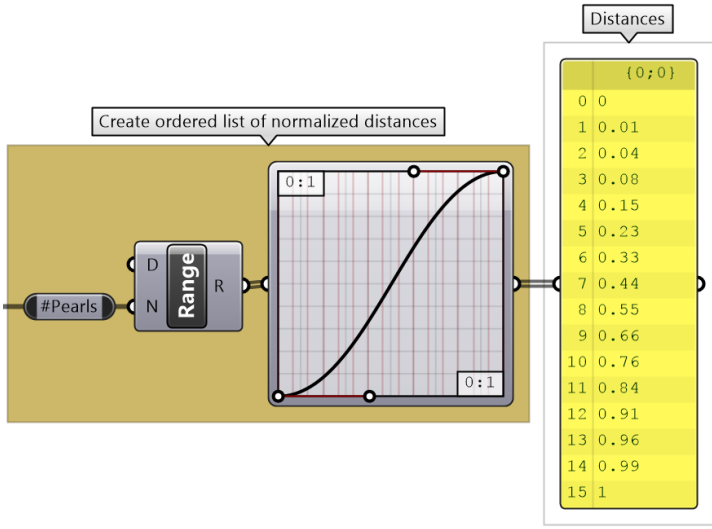
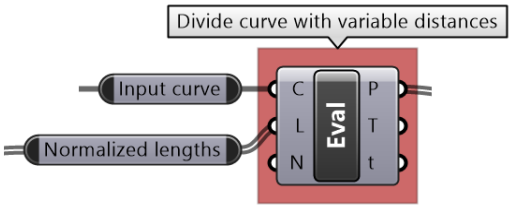
Pattern para seleccionar los puntos (seleccione ***invert*** para la entrada del patrón).	
Proceso intermedio 4 Para crear puntos intermedios, utilice el componente ***Weave*** en los puntos superiores e inferiores.	
Junte todos los procesos 	

2_5_4: Tutorial de un collar de perlas

Cree un collar con una perla grande en el centro y, gradualmente, perlas de menor tamaño hacia los extremos, como en la imagen. Haga que el número de perlas sea paramétrico entre 15-25.



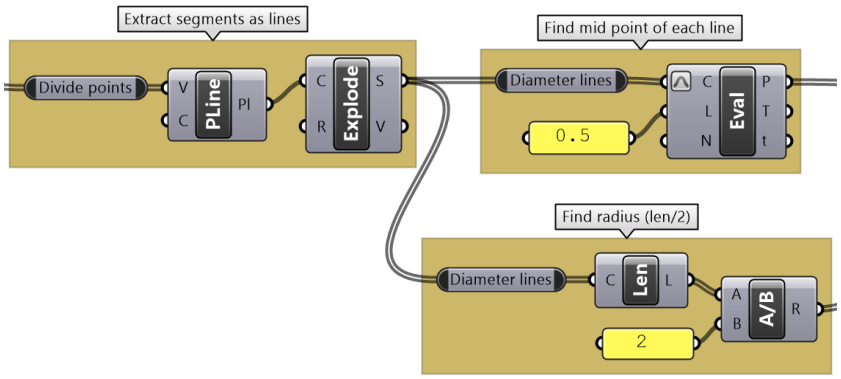
Análisis del algoritmo	
El flujo de trabajo para crear el collar sigue estas líneas generales: 1- Divida la curva en segmentos de distancias variables (más anchos en el centro y más estrechos hacia los extremos). 2- Halle los puntos medios de cada segmento y su longitud. 3- Cree esferas en los centros usando la mitad de la longitud como radio.	

Pasos de la solución	
Salida: Las superficies	
Proceso clave: Utilice el componente ***Sphere*** para generar las superficies.	
Dada la entrada: Curva del collar y Número de perlas como parámetro (el usuario puede cambiarlo)	
Proceso intermedio 1: El componente ***Range*** crea distancias iguales. Necesitamos cambiar a distancias variables y podemos utilizar el componente ***Graph Mapper*** para controlar el espaciado.	
Proceso intermedio 2: Puesto que hemos normalizado las distancias desde el inicio de la curva (los parámetros están entre 0 y 1), podemos utilizar el componente ***Evaluate Length*** para hallar los puntos de división.	

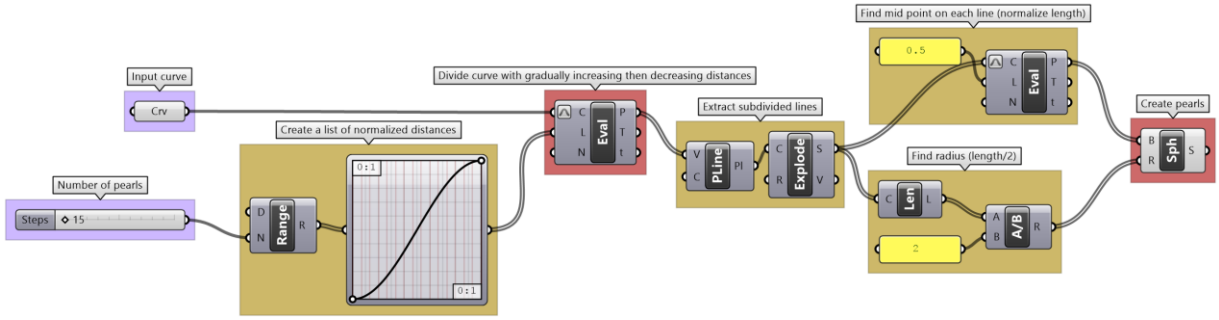
Proceso intermedio 3:
Genere los segmentos. Utilice los componentes ***Polyline*** y ***Explode*** para convertir los puntos en segmentos.

Los puntos centrales se calculan en la mitad de los segmentos. Utilice ***Evaluate Length*** en la mitad.

El radio es la mitad de la longitud de cada segmento. Utilice los componentes ***Length*** y ***Division***.



Junte todos los procesos



Capítulo 3: Estructuras de datos avanzadas

Este capítulo presenta la estructura de datos avanzada en GH, es decir, los árboles de datos y a las diferentes formas de generarlos y gestionarlos. El objetivo es saber cuándo y cómo utilizar las estructuras de árbol, así como las mejores prácticas para utilizarlas y manipularlas con eficacia.

3_1: La estructura de datos de Grasshopper

3_1_1: Introducción

En programación, hay muchas estructuras de datos que rigen la forma de almacenar y acceder a los datos. Las estructuras de datos más comunes son las variables, las matrices y las matrices anidadas. Existen otras estructuras de datos optimizadas para fines específicos, como la clasificación o la minería de datos. En Grasshopper, solo hay una estructura para almacenar datos, el **árbol de datos**. ¿Qué pasa con lo que hemos aprendido hasta ahora: **elemento único** y **lista de elementos**? En GH, esos se trata simplemente de árboles. Un elemento único es un árbol con una rama, que tiene un elemento, y una lista es un árbol con una rama que tiene varios elementos. Suena bien poder encajar todos los datos en una estructura de datos unificadora, pero esto requiere que el usuario sea consciente y esté atento a cómo cambia su estructura de datos entre operaciones y cómo esto puede afectar a los resultados deseados. Este capítulo intenta desmitificar el árbol de datos de Grasshopper.

3_1_2: Procesamiento de árboles de datos

Hemos utilizado los componentes ***Panel*** y ***Parameter Viewer*** para ver la estructura de datos. Utilizaremos ambos bastante para mostrar cómo se almacenan los datos. Empecemos con una entrada de un elemento único. El ***Parameter Viewer*** tiene dos modos de visualización, uno es con texto y el otro es gráfico. Puede ver que la entrada de elemento único se almacena en una rama que solo tiene un elemento.

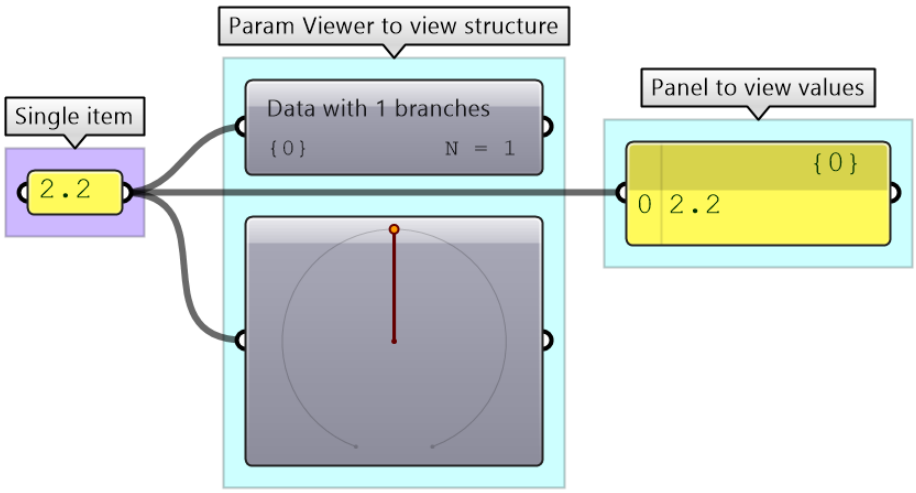


Figura (51): Diferentes formas de previsualizar la estructura de datos en Grasshopper

El ***Parameter Viewer*** muestra cada dirección de rama (denominada "Ruta") y el número de elementos de esa rama, como se muestra en la Figura (52).

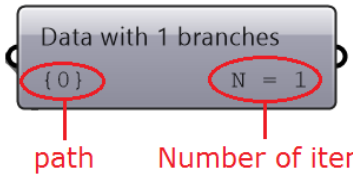


Figura (52): El ***Parameter Viewer*** indica la dirección de la ruta y el número de elementos de cada rama

Un árbol con una rama normalmente almacena una lista de elementos. Figura (53). Sin embargo, los tres elementos también pueden almacenarse en tres ramas diferentes. Figura (54).

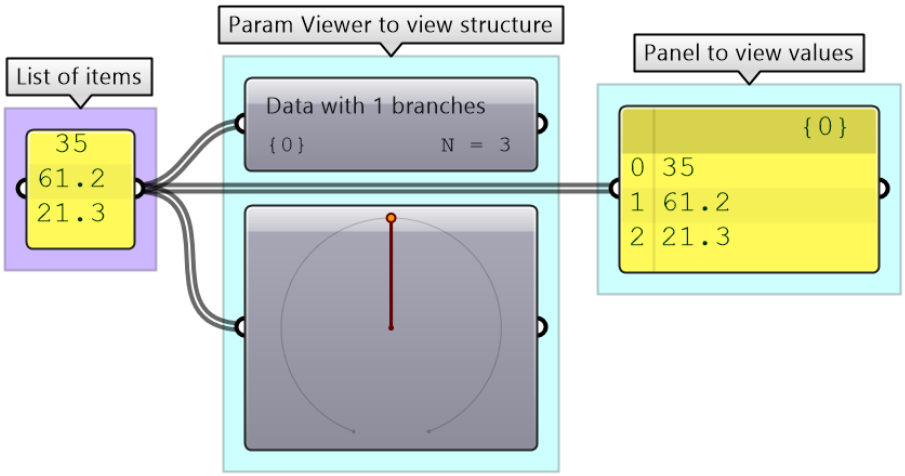


Figura (53): Una lista es un árbol que tiene una rama con varios elementos

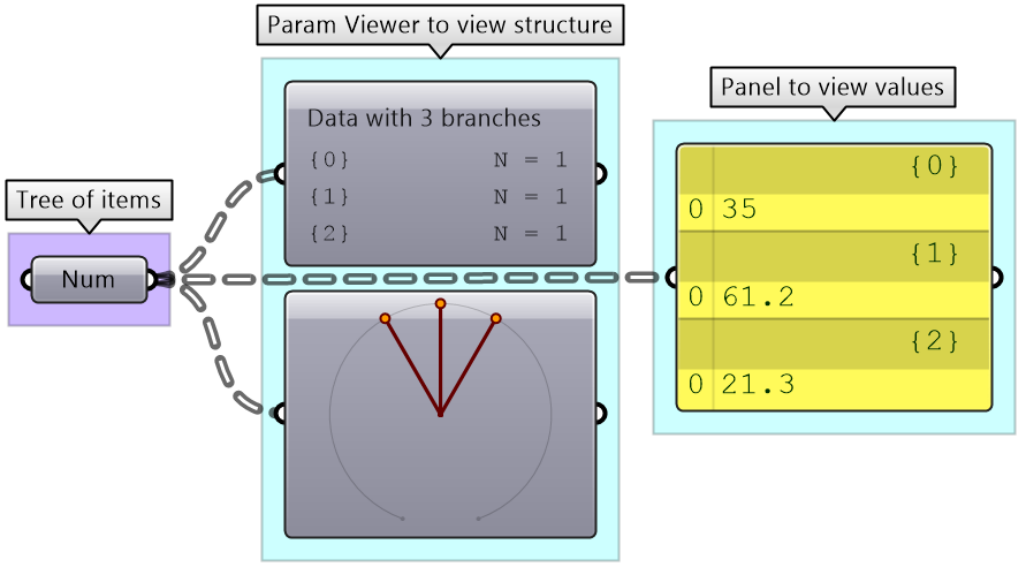


Figura (54): Un árbol contiene cualquier número de ramas con cualquier número de elementos en cada rama

La clave para entender la estructura de datos de Grasshopper es poder responder a la siguiente pregunta: **¿cuál es la diferencia de almacenar los 3 números en una rama en lugar de tres ramas?**

La estructura de datos informa a los componentes de GH sobre cómo hacer coincidir los valores de entrada. En otras palabras, los componentes pueden procesar los datos de forma diferente en función de su estructura. El siguiente ejemplo ilustra cómo diferentes estructuras de datos para el mismo conjunto de valores pueden afectar al resultado.

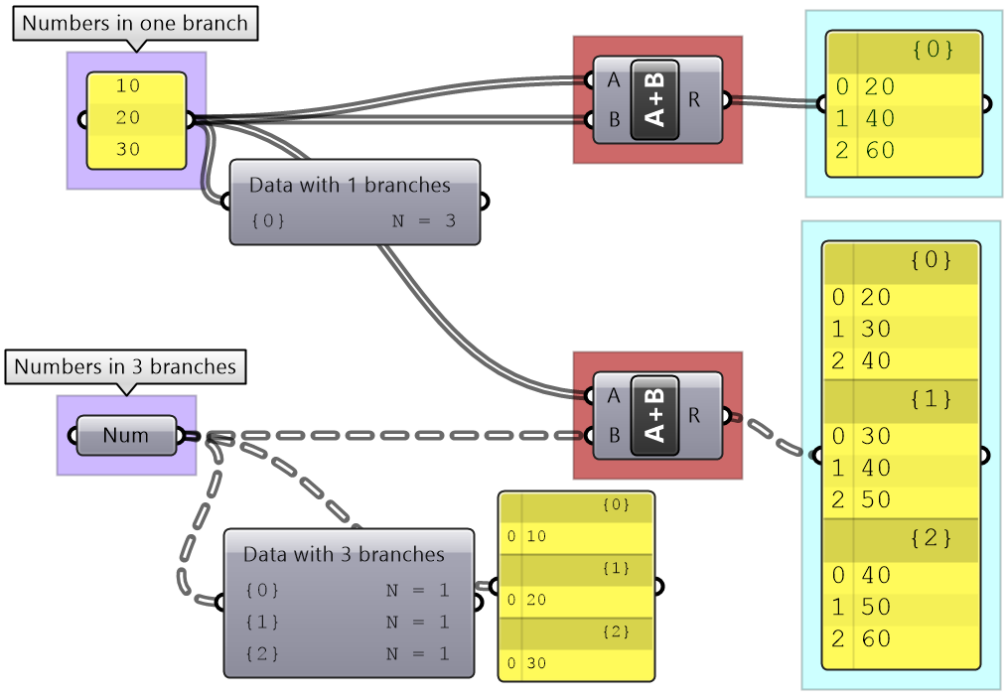


Figura (55): Organizar el mismo conjunto de valores en diferentes estructuras de datos afecta a la salida

Más adelante explicaremos la correspondencia de árboles de datos, pero ya puede ver que los componentes de GH tienen en cuenta la estructura de datos y el resultado puede variar considerablemente en función de la misma. Esta es una de las complicaciones heredadas al utilizar una estructura de datos unificadora en un lenguaje de programación.

3_1_3: Notación de árboles de datos

El primer paso para entender los árboles de datos es aprender la notación de los árboles de GH. Como hemos visto en los ejemplos, los árboles están formados por ramas, y cada rama contiene un número determinado de elementos. La dirección o ruta de cada rama se representa con números enteros separados por punto y coma y entre corchetes. El índice de cada elemento está entre corchetes. Este diagrama muestra un desglose de la dirección de los elementos en los árboles.

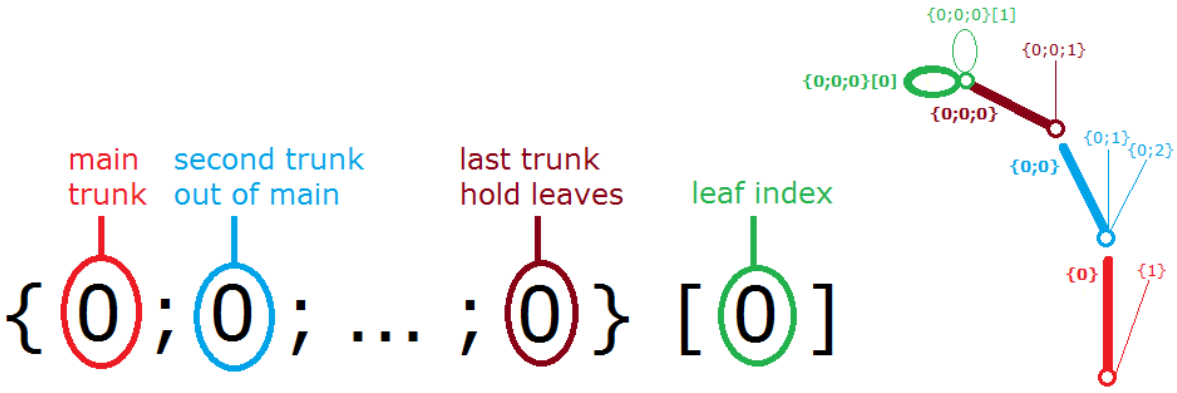


Figura (56): La dirección de los elementos incluye la dirección de la rama y el índice del elemento de la rama

Estos son algunos ejemplos de varias estructuras de árboles y cómo se muestran en el ***Parameter Viewer*** y el ***Panel***.

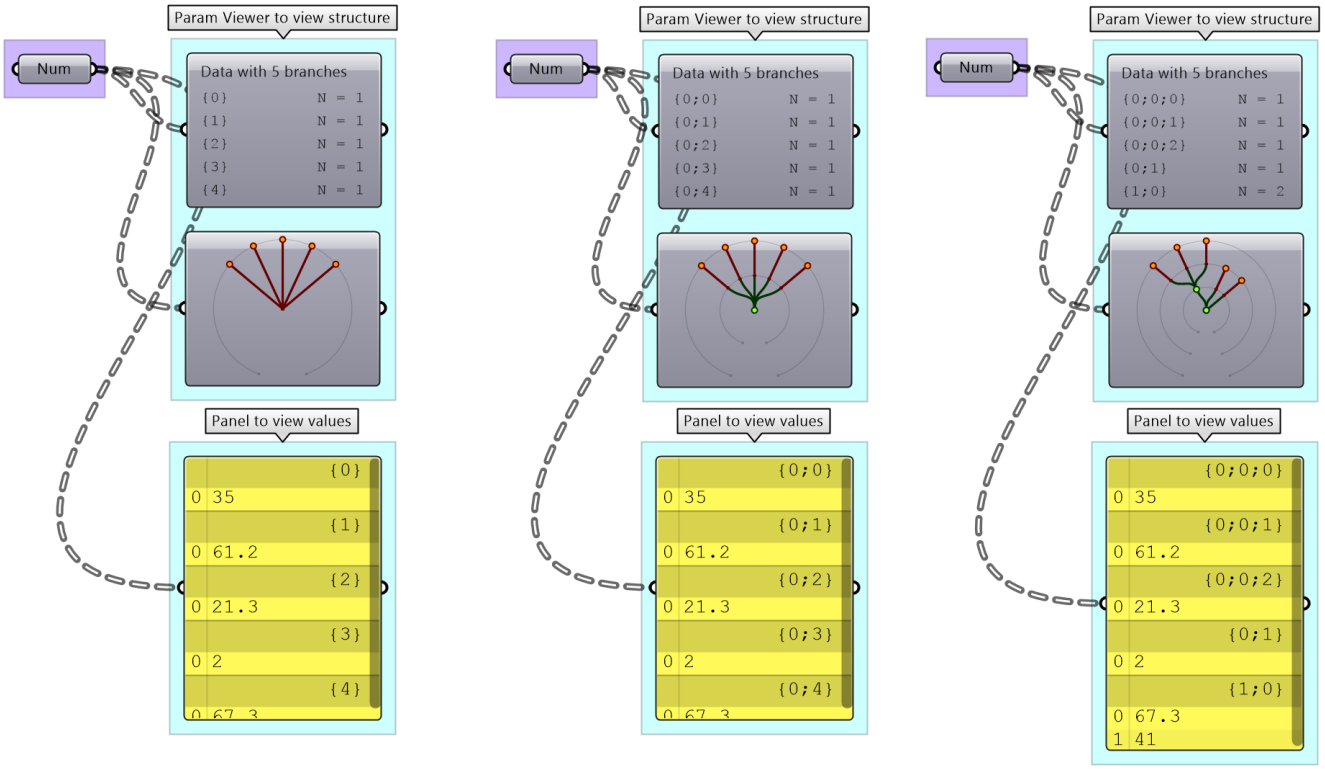
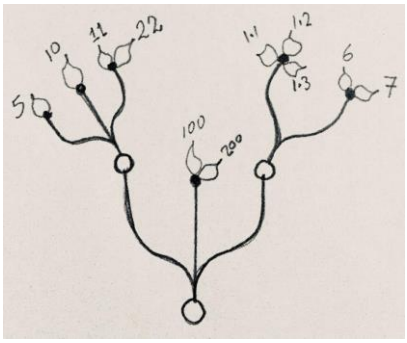


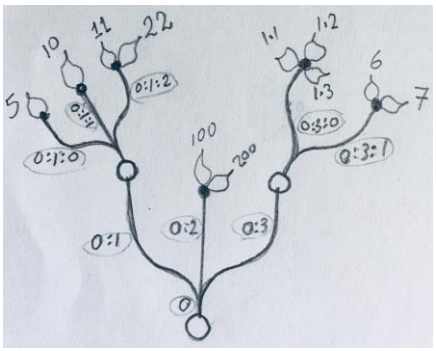
Figura (57): El mismo conjunto de valores se mantiene en diferentes estructuras.
Izquierda: 5 troncos (5 árboles) con un elemento en cada uno. Medio: 5 ramas que salen de un tronco (1 árbol) y cada rama contiene un único objeto. Derecha: dos troncos (2 árboles), el primero tiene 2 ramas y una de ellas se ramifica en 3 ramas, cada una de las cuales tiene un elemento, y el segundo tiene un elemento. El segundo tronco tiene 2 elementos.

3_1_4: Tutorial de árbol de datos

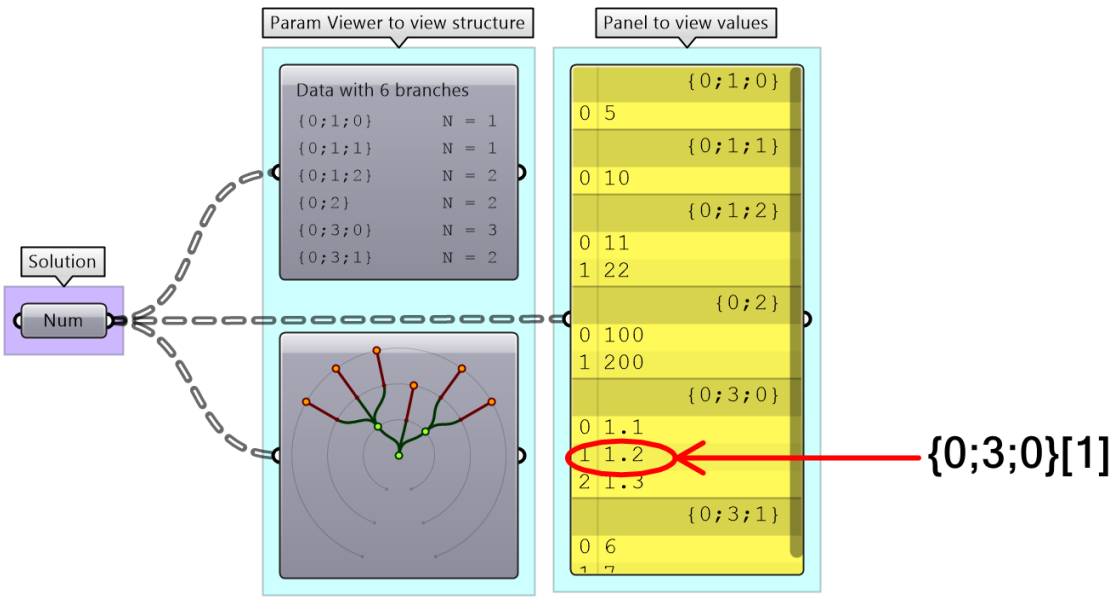
Cree un árbol de números utilizando el parámetro ***Number***, que se parece a la imagen cuando se ve en el ***Param Viewer*** y contiene los números indicados. A continuación, escriba en un ***Panel*** la dirección completa al elemento "1.2" . Tenga en cuenta que el orden de las ramas y las hojas es siempre de izquierda a derecha en el sentido de las agujas del reloj.



Solución



La ruta de “**1.2**” es: **{ 0 ; 3 ; 0} [1]**
Nota: las tres ramas del tronco principal se definen aquí en 0:1, 0:2 y 0:3. También podrían haber sido 0:0, 0:1 y 0:2. Ambas opciones son correctas.



3_2: Generación de árboles

Hay muchas formas de generar árboles de datos complejos. Algunas son explícitas, pero la mayoría son el resultado de algunos procesos, por lo que siempre hay que conocer las estructuras de datos de la salida antes de utilizarlas como entrada. Es posible introducir los datos y definir la estructura de datos directamente dentro de cualquier parámetro de Grasshopper. Una vez definida, es relativamente difícil de cambiar, por lo que va mejor para una entrada constante. A continuación se muestra un ejemplo de cómo definir el árbol de datos directamente dentro de un parámetro.

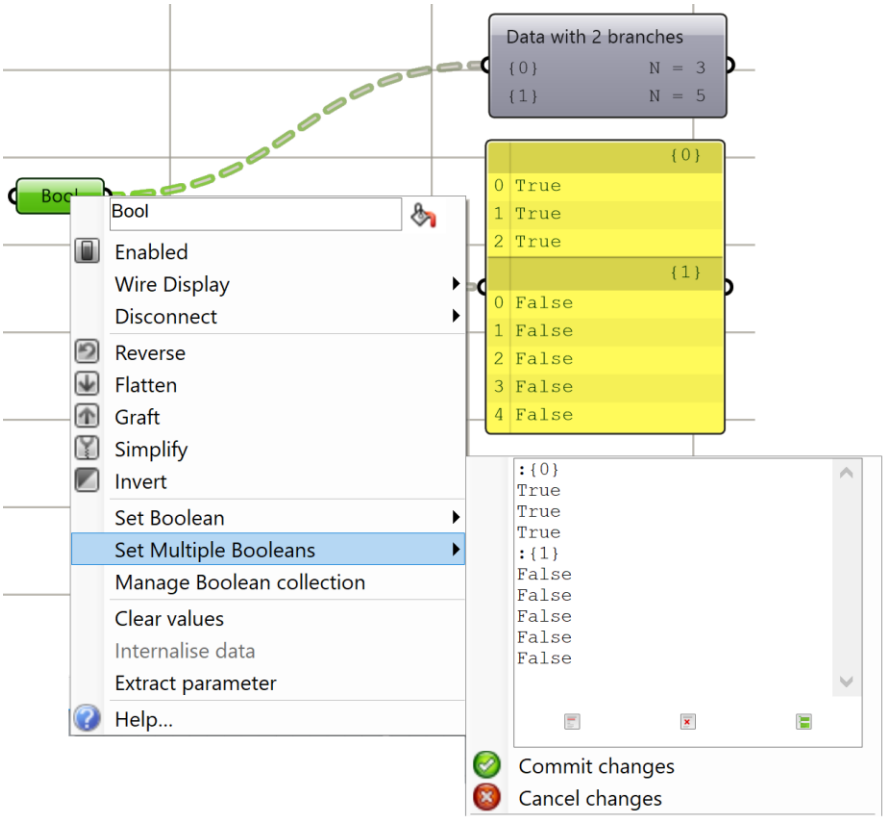
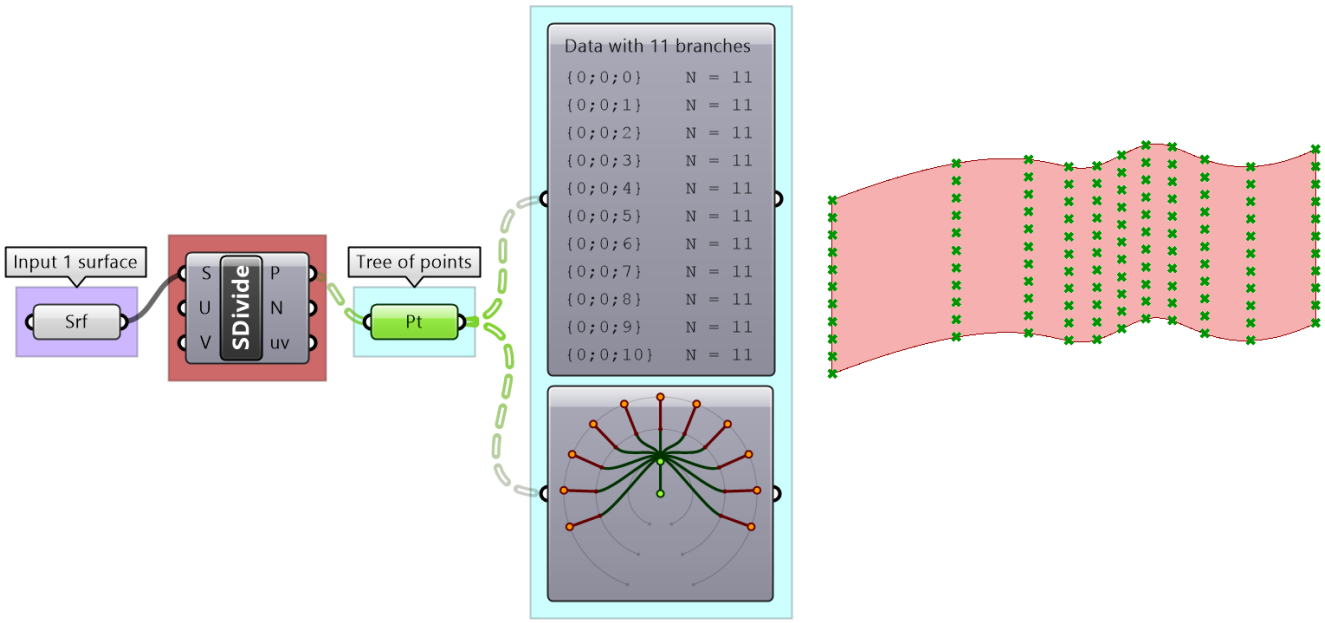


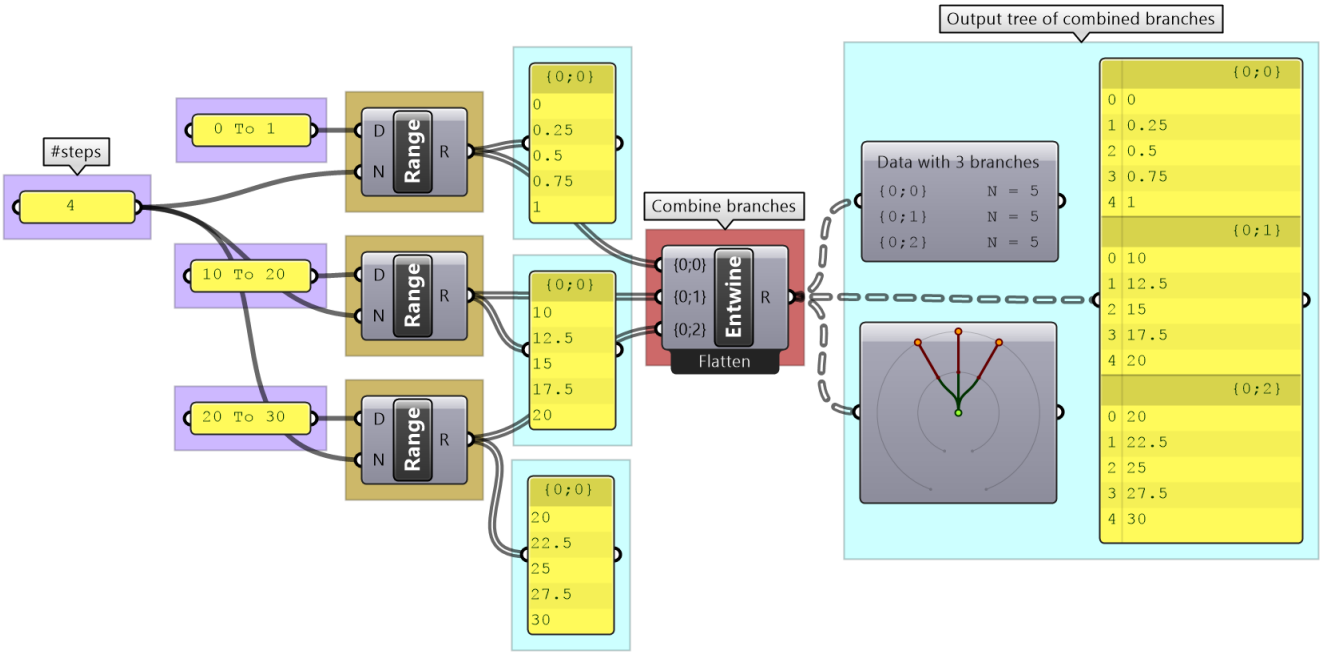
Figura (58): Definir árboles de datos directamente dentro del parámetro

Hay muchos componentes que generan árboles de datos, como ***Grid*** y ***DivideSrf***, y otros que combinan listas en una estructura de árbol, como ***Entwine***. Además, todos los componentes que producen listas también pueden crear árboles si la entrada es una lista. Por ejemplo, si introducimos más de una curva en el componente ***DivideCrv*** , obtendremos un árbol de puntos.



Figura(59): El componente ***SDivide*** toma una entrada (superficie) y produce un árbol de datos (malla) como salida.

Todos los componentes que generan listas de números (como ***Range*** y ***Series***) también pueden generar árboles cuando se proporciona una lista de entrada.



Figura(60): El componente ***Entwine*** toma un número determinado de listas y las combina en una estructura de árbol.

Quizás uno de los casos más comunes para generar un árbol es cuando se divide una lista de curvas para generar una rejilla de puntos. De modo que la entrada es una lista y la salida es un árbol.

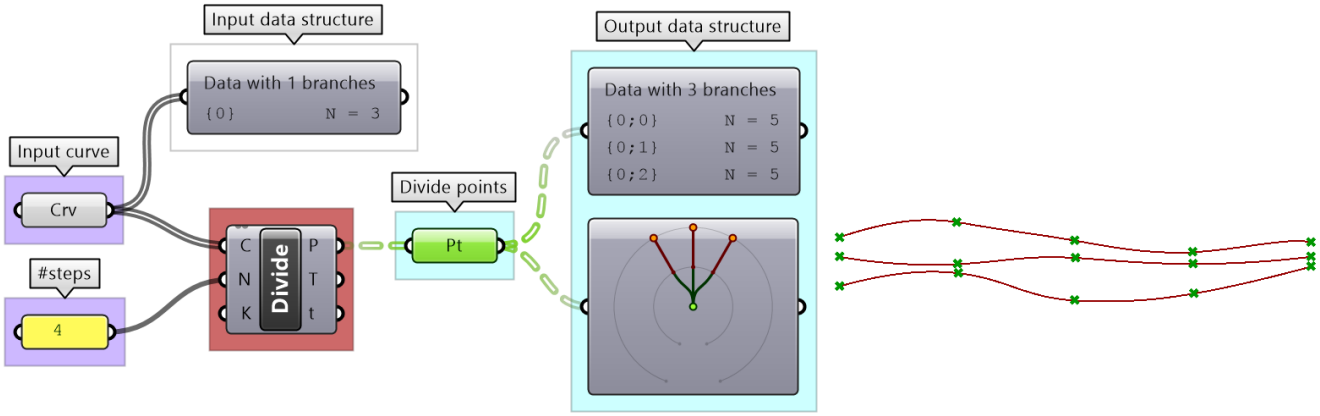
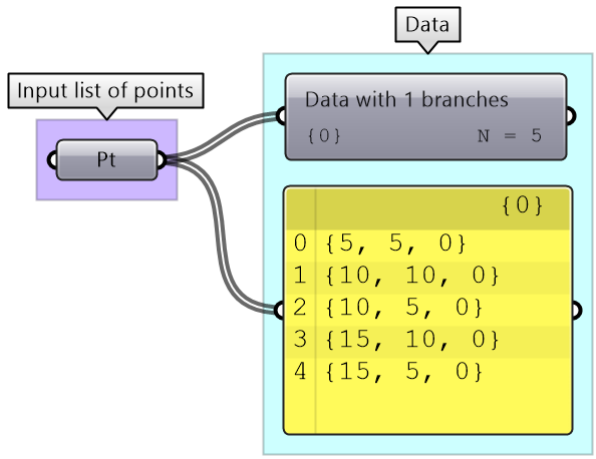


Figura (61): El componente ***Divide*** toma cualquier lista (curvas) y genera una estructura de árbol (rejilla).

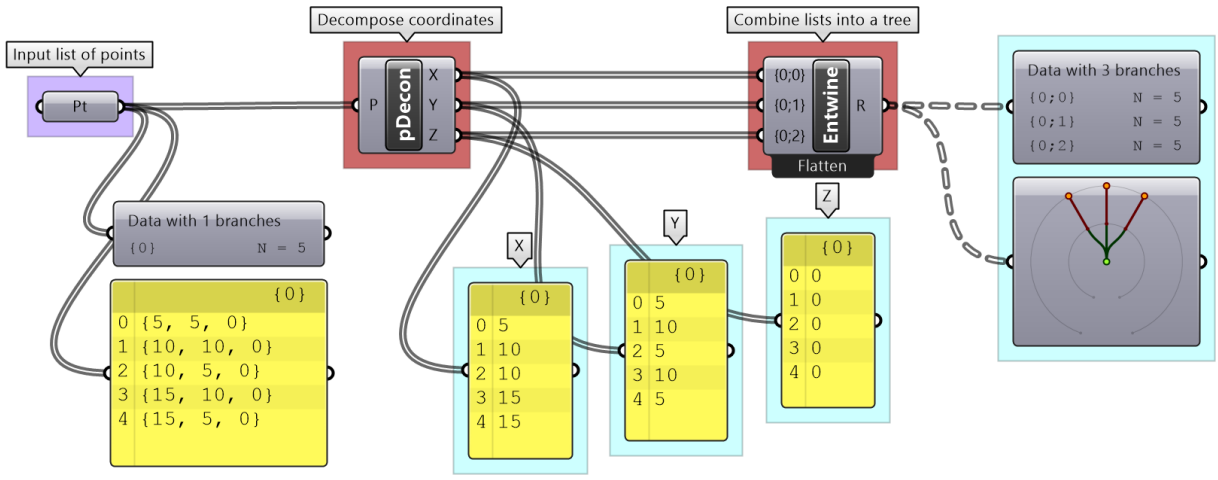
3-2-1 Tutorial de generación de árboles

Dada la siguiente lista de puntos, cree un árbol numérico con 3 ramas, una para cada coordenada.



Solución

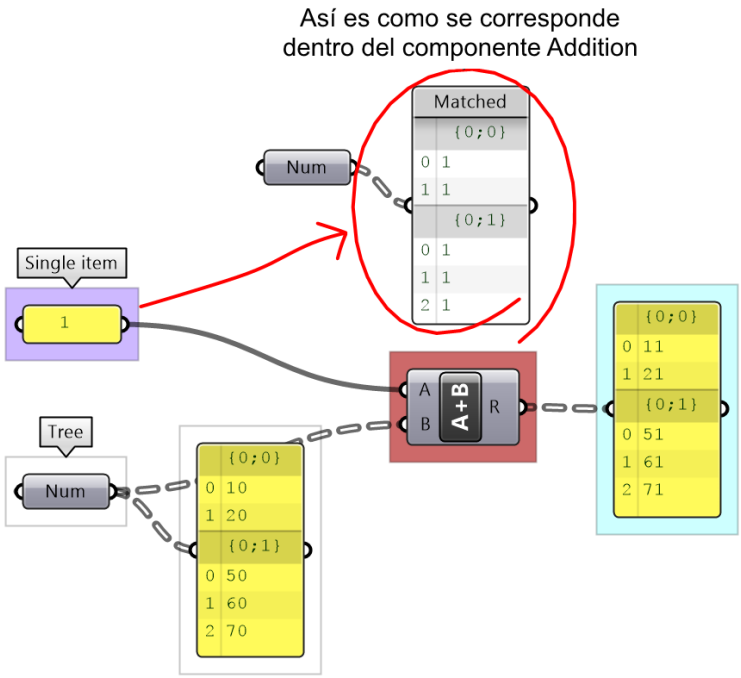
Explicación: cada punto de entrada es un elemento con un dato único que contiene 3 números (coordenadas). Queremos aislar cada coordenada en una lista separada y luego combinarlas en una estructura de datos. Por lo tanto, primero tenemos que deconstruir los puntos de entrada (utilice *Deconstruct* del componente *pDecon*) y luego combinar las listas en una sola estructura (utilice el componente *Entwine*).



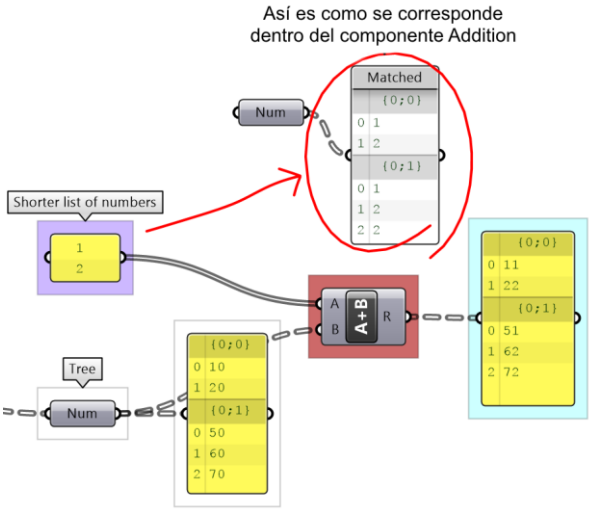
3_3: Correspondencia de árboles

Hemos explicado la correspondencia de listas ***largas***, ***cortas*** y ***cruzadas***. Los árboles siguen convenciones similares para ampliar las ramas más cortas repitiendo el último elemento coincidente. Si un árbol tiene menos ramas, se repite la última rama. A continuación se muestran algunos casos de correspondencia de árboles.

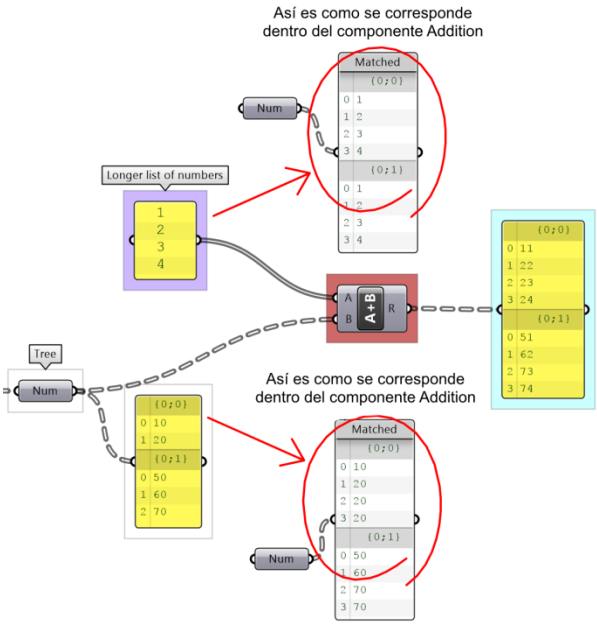
Hacer corresponder un objeto con un árbol:



Hacer corresponder una lista más corta con un árbol (ramas del árbol más largas que la lista):

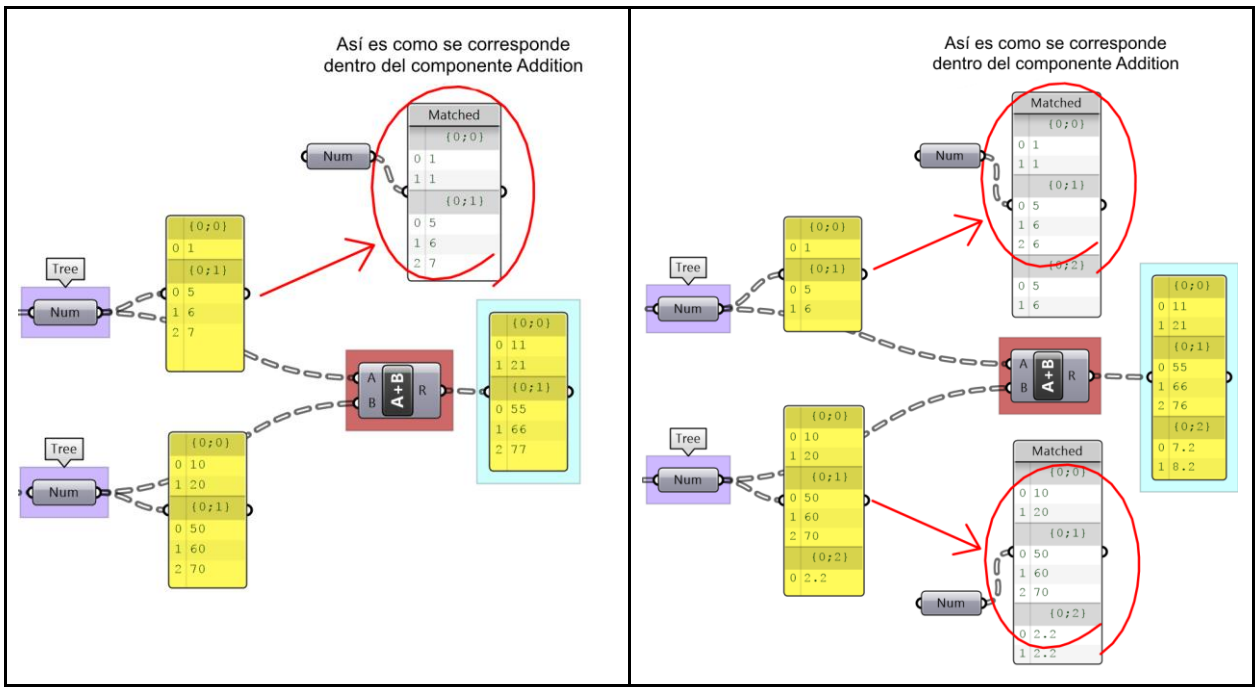


Hacer corresponder una lista más larga con un árbol (ramas del árbol más cortas que la lista):



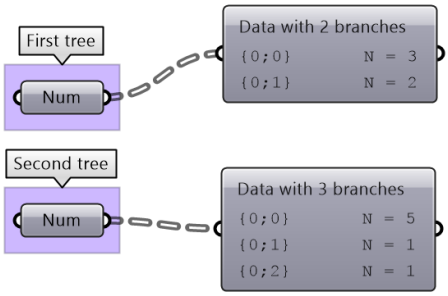
Hacer corresponder 2 árboles con el mismo número de ramas:

Hacer corresponder 2 árboles con diferente número de ramas:



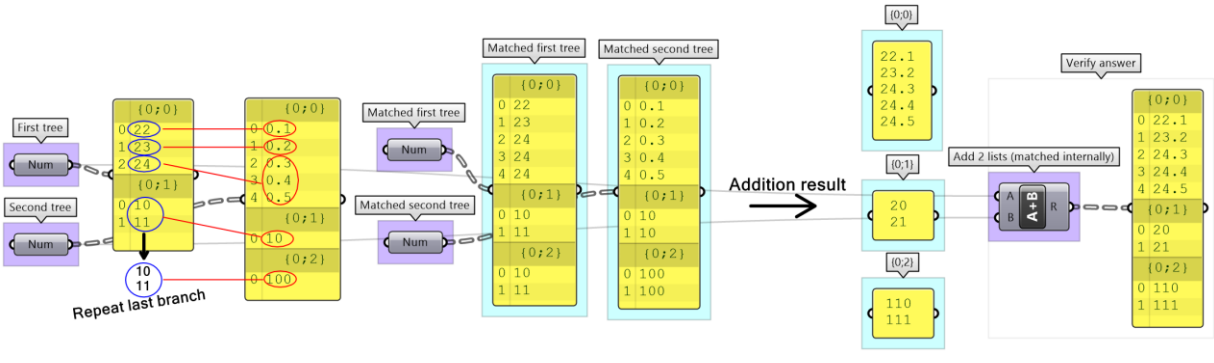
3_3_1 Tutoriales de correspondencia de árboles

Examine las 2 estructuras numéricas siguientes y prediga la estructura y el resultado de sumarlas (con la coincidencia predeterminada de Grasshopper). Verifique su respuesta utilizando los componentes de ***Addition***.



Solución

Solución clave: Los dos árboles de entrada tienen diferente número de ramas y diferente número de elementos en cada rama. La última rama del árbol más corto se repite para igualar el número de ramas, y luego se igualan las ramas correspondientes repitiendo el último elemento de la rama más corta.



3_4: Cruzar árboles

Grasshopper proporciona componentes para ayudar a extraer ramas y elementos de los árboles. Si tiene la ruta a una rama o a un elemento, puede utilizar los componentes ***Branch*** y ***Item***. Debe comprobar la estructura de su entrada para poder ofrecer la ruta correcta.

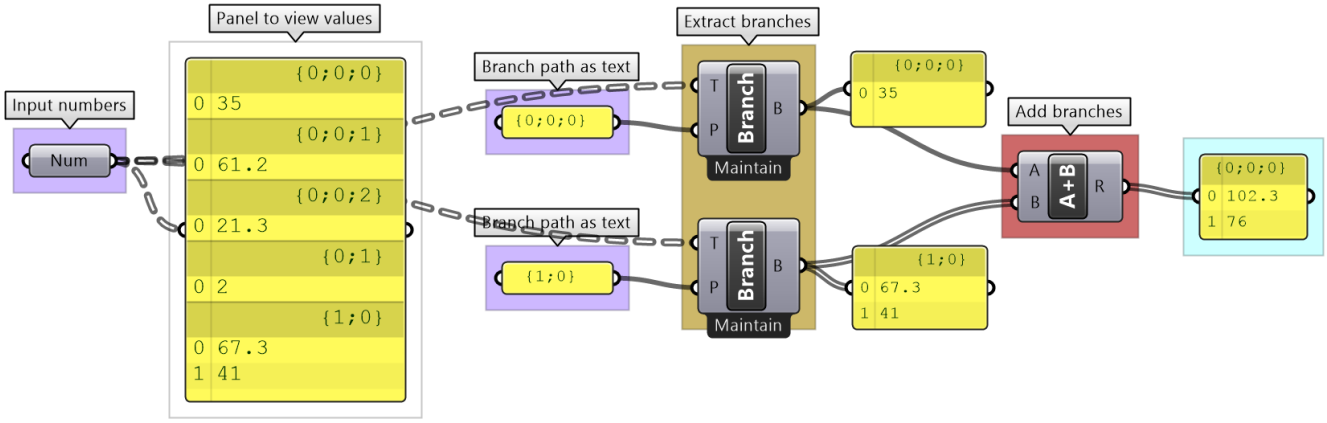


Figura (62): Seleccione ramas de un árbol

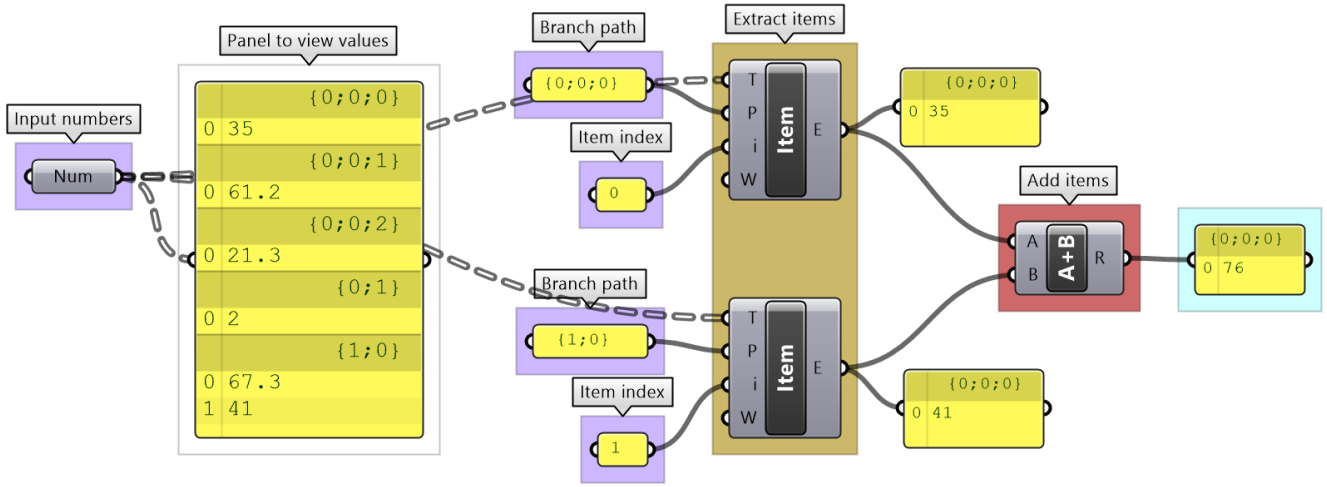


Figura (63): Seleccione elementos de un árbol

Si sabe que su estructura puede cambiar, o simplemente no quiere escribir la ruta, puede extraerla utilizando los componentes ***Param Viewer*** y ***List Item***.

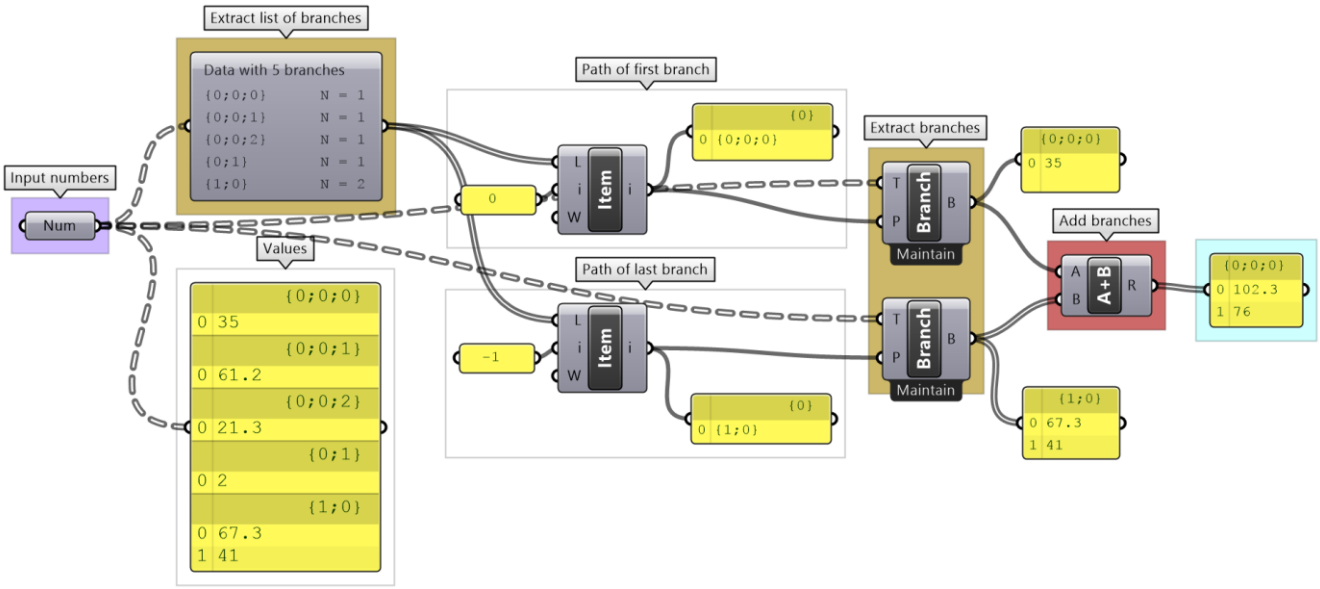
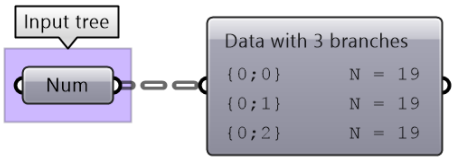


Figura (64): Ejemplo de cómo extraer rutas de datos dinámicamente

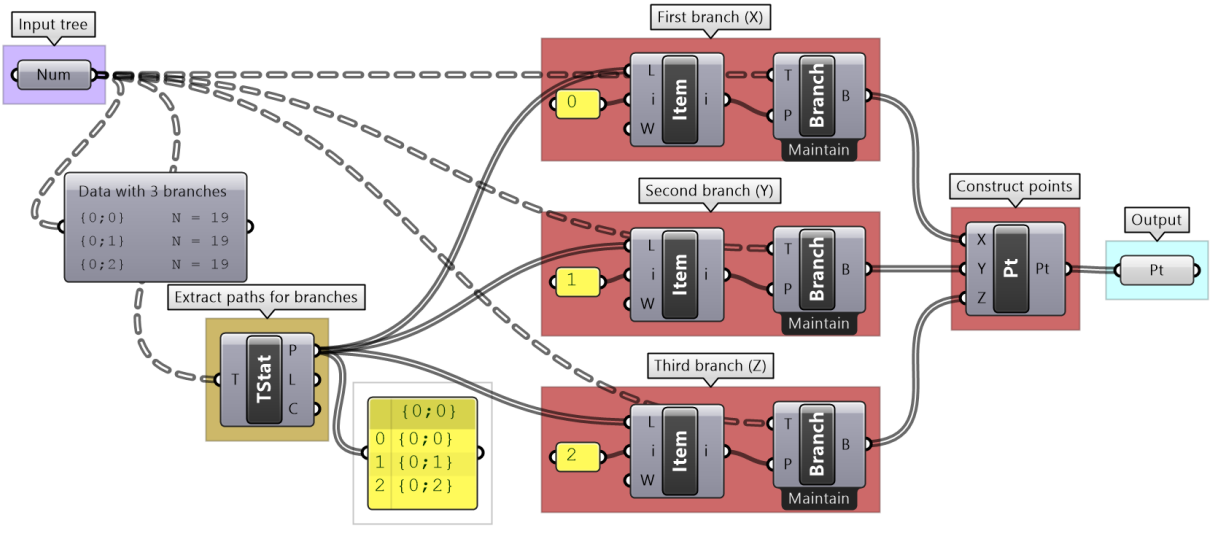
3_4_1 Tutorial para cruzar árboles

El siguiente árbol tiene 3 ramas para cada una de las coordenadas (x, y, z) de alguna lista de puntos. Utilice ese árbol para crear una lista con estos puntos.



Solución

Solución clave: Podemos crear una lista de puntos utilizando como entrada 3 listas que representan los valores X, Y y Z. Si podemos aislar las 3 ramas del árbol de entrada, entonces podremos conectarlas con el componente de construcción de puntos.



3_5: Operaciones básicas con árboles

Las operaciones básicas con árboles son muy usuales y puede que las necesite en la mayoría de soluciones. Es muy importante entender qué hacen estas operaciones y cómo afectan a la salida.

3_5_1: Visualizar la estructura del árbol

Como hemos visto en la correspondencia de datos, diferentes estructuras de datos del mismo conjunto de elementos producen resultados distintos. Grasshopper ofrece tres formas de ver la estructura de datos, el ***Parameter Viewer*** en formato de texto o diagrama, y el ***Panel***.

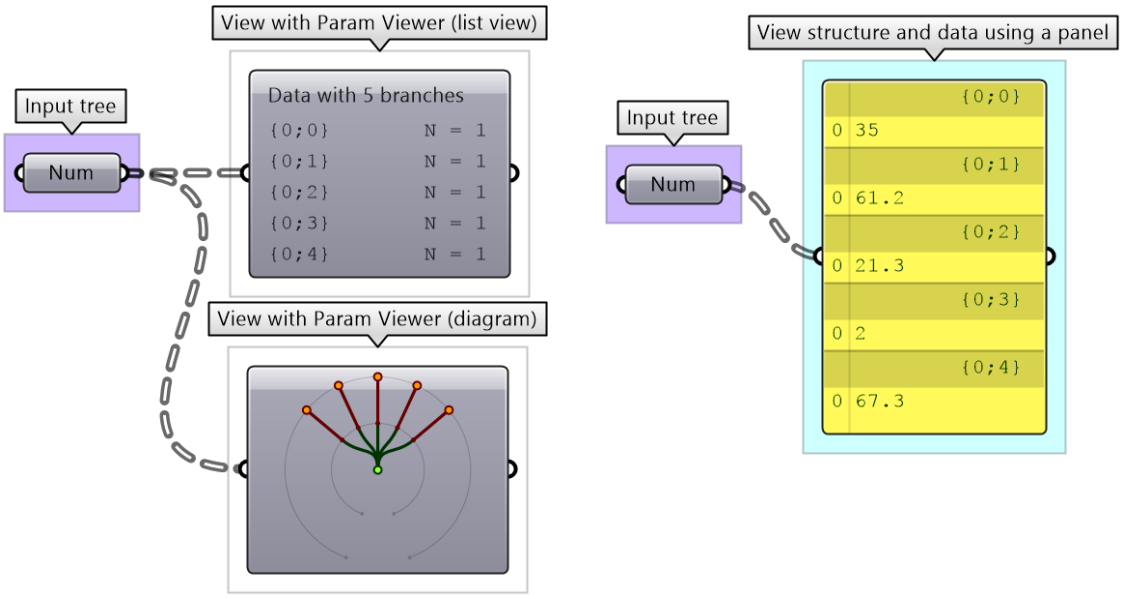


Figura (65): Visualización de árboles con los componentes ***Parameter Viewer*** y ***Panel***

La información del árbol se puede extraer con el componente TStats. Puede extraer la lista de rutas de todas las ramas, el número de elementos de cada rama y el número de ramas.

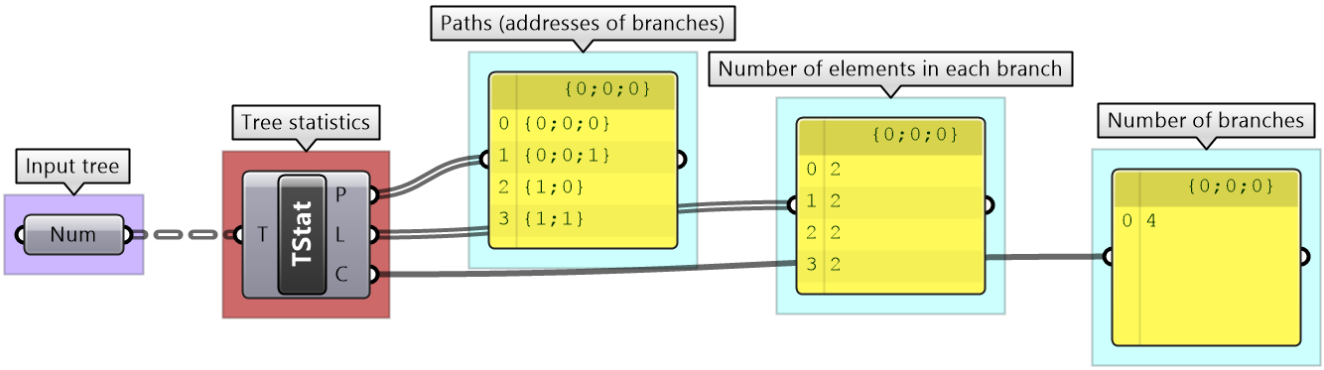
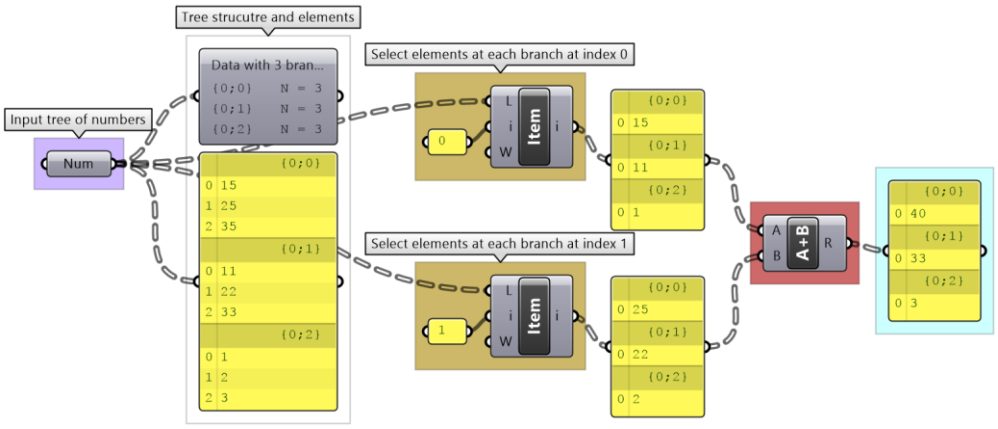
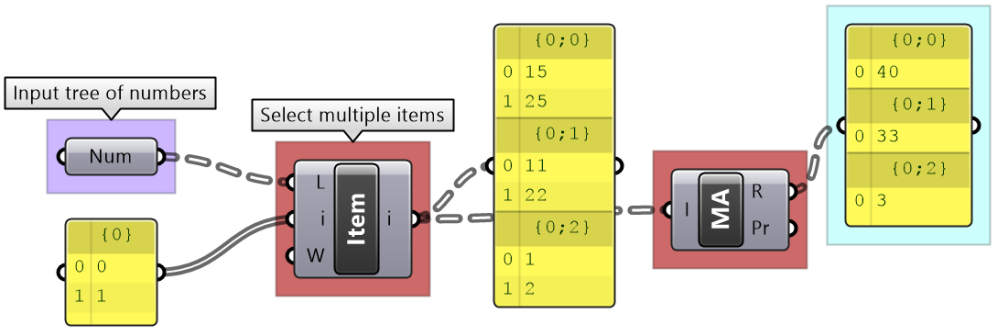
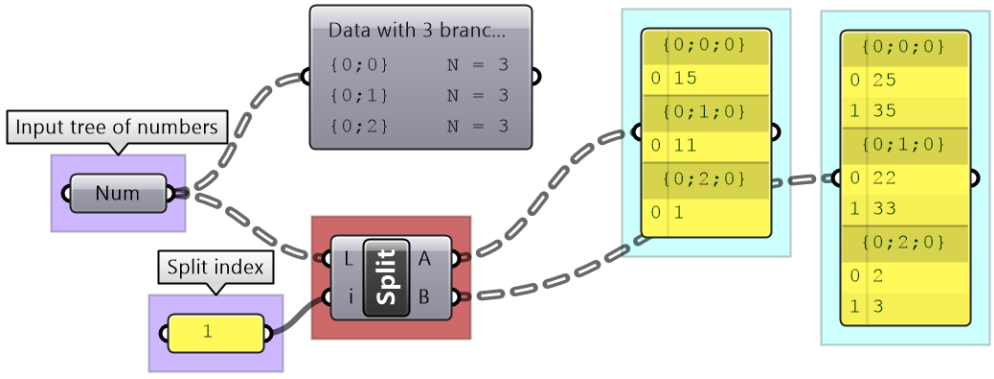


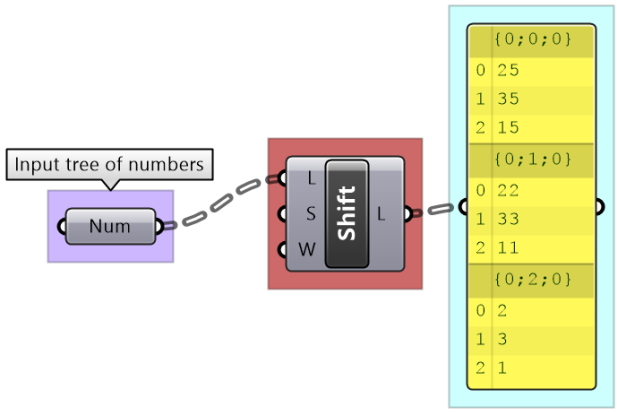
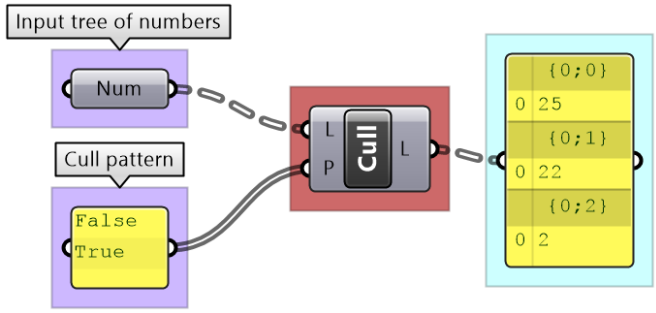
Figura (66): Extracción de la estructura de árboles con el componente ***TStats***

3_5_2: Operaciones de lista con árboles

Los árboles pueden considerarse como una lista de ramas. Cuando se utilizan operaciones de lista en árboles, cada rama se trata como una lista separada y la operación se aplica a cada rama de forma independiente. Es difícil predecir la estructura de datos resultante, por lo que siempre es importante comprobar las estructuras de entrada y salida antes y después de aplicar cualquier operación.

Para ilustrar cómo funcionan las operaciones de lista en árboles, utilizaremos un árbol sencillo, una rejilla de puntos y aplicaremos diferentes operaciones de lista sobre el árbol. A continuación, examinaremos la salida y su estructura de datos.

Operaciones	Ejemplo de cómo se aplica la operación de lista a los árboles
List Item Seleccione los elementos en un índice específico de cada rama	
List Item Seleccione varios índices para aislar una parte del árbol y realizar una operación, por ejemplo ***Mass Addition***	
Split List Divide los elementos de las ramas en 2 árboles en el índice especificado	

Shift List Cambia los elementos de cada rama	
Cull Pattern Oculta cada rama	

3_5_3: Insertar de listas en árboles

En algunos casos es necesario convertir una lista en un árbol en el que cada elemento se coloca en su propia rama. La inserción puede utilizar árboles complejos con ramas de profundidad variable.

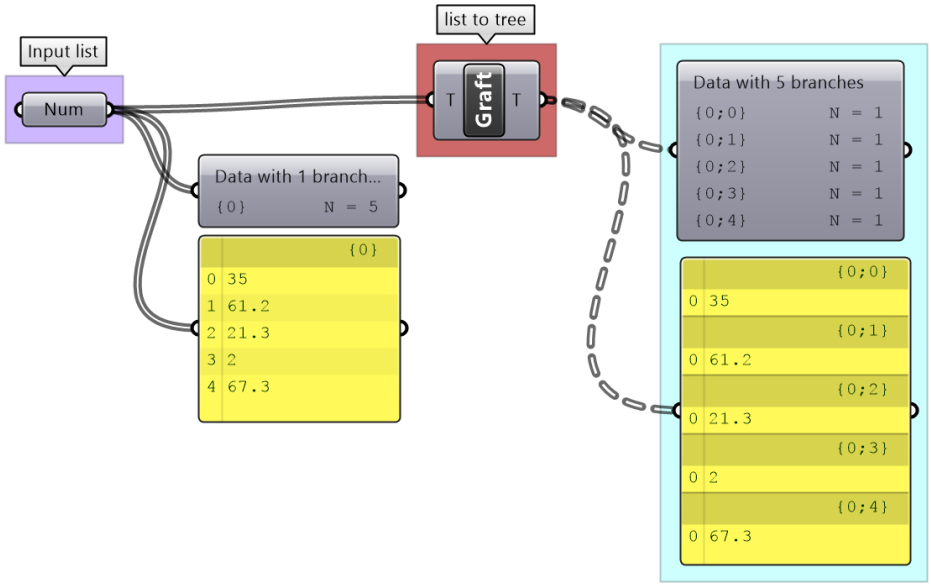


Figura (67): La inserción de un árbol crea una nueva rama para cada elemento

Puede parecer poco intuitivo complicar la estructura de datos (de una lista simple a un árbol), pero las inserciones son muy útiles para intentar conseguir ciertas correspondencias. Por ejemplo, si necesita sumar cada elemento de una lista con todos los elementos de la segunda lista, tendrá que insertar la primera lista antes de introducirla en el proceso de suma.

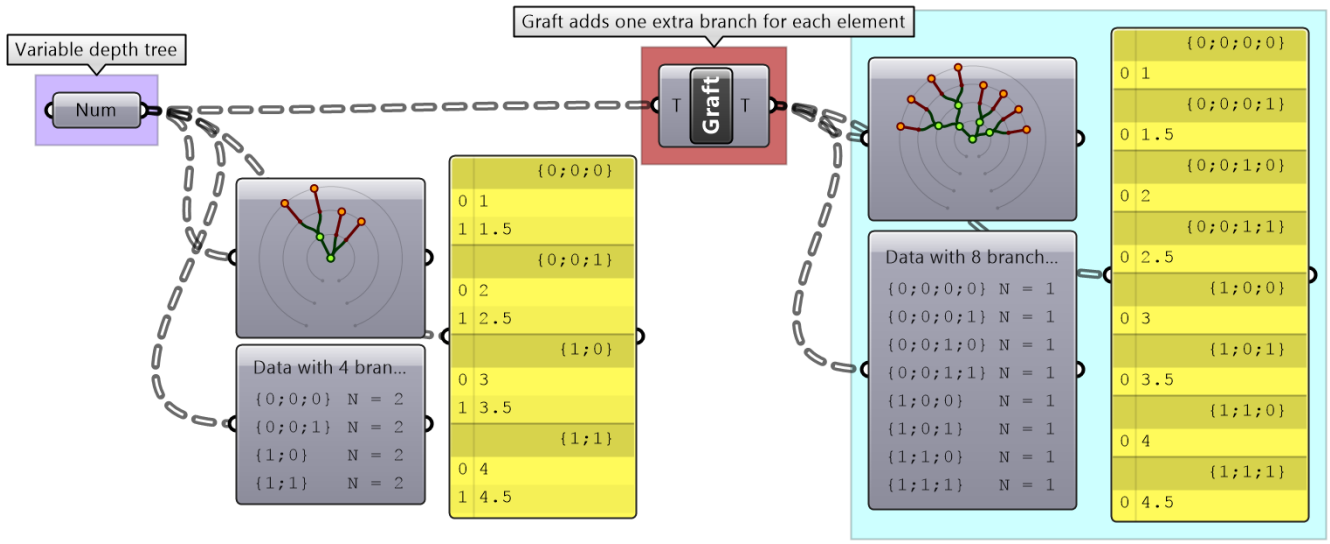


Figura (68): Inserción de árboles complejos

3_5_4: Aplanar de árboles a listas

Otras veces puede ser necesario convertir la estructura de árbol en una simple lista. Esto se logra con el proceso de aplanamiento. Los datos de cada rama se extraen y se añaden secuencialmente a una lista.

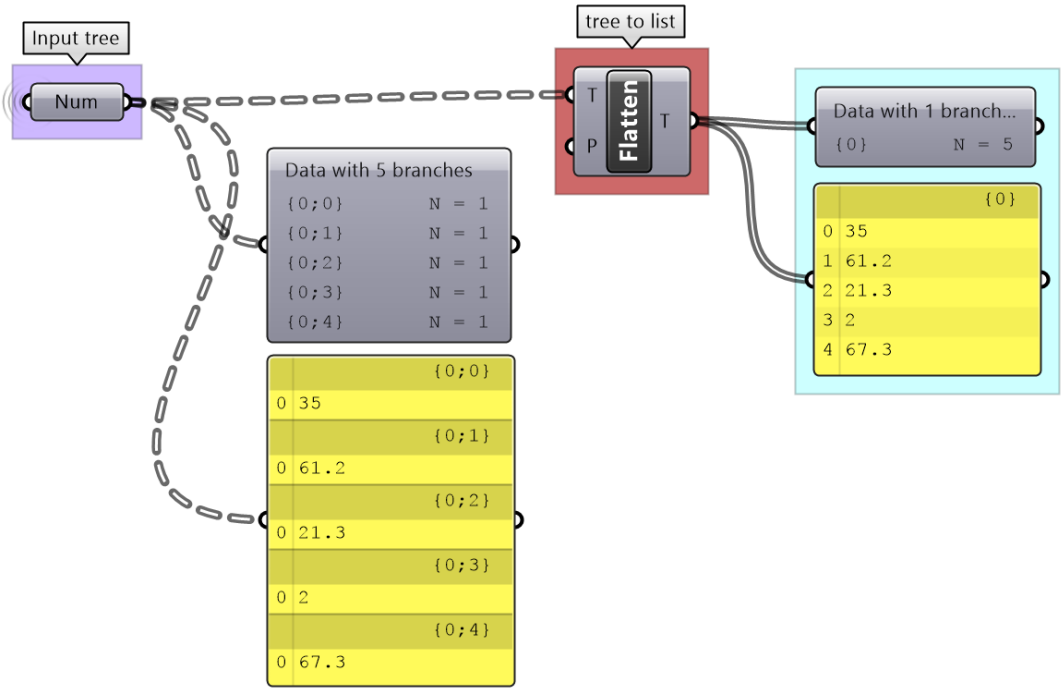


Figura (69): El aplanamiento coloca todos los elementos del árbol en una lista

Aplanar también puede utilizar cualquier árbol complejo. Coge las ramas en orden empezando por el tronco de menor índice y pone todos los elementos en una lista.

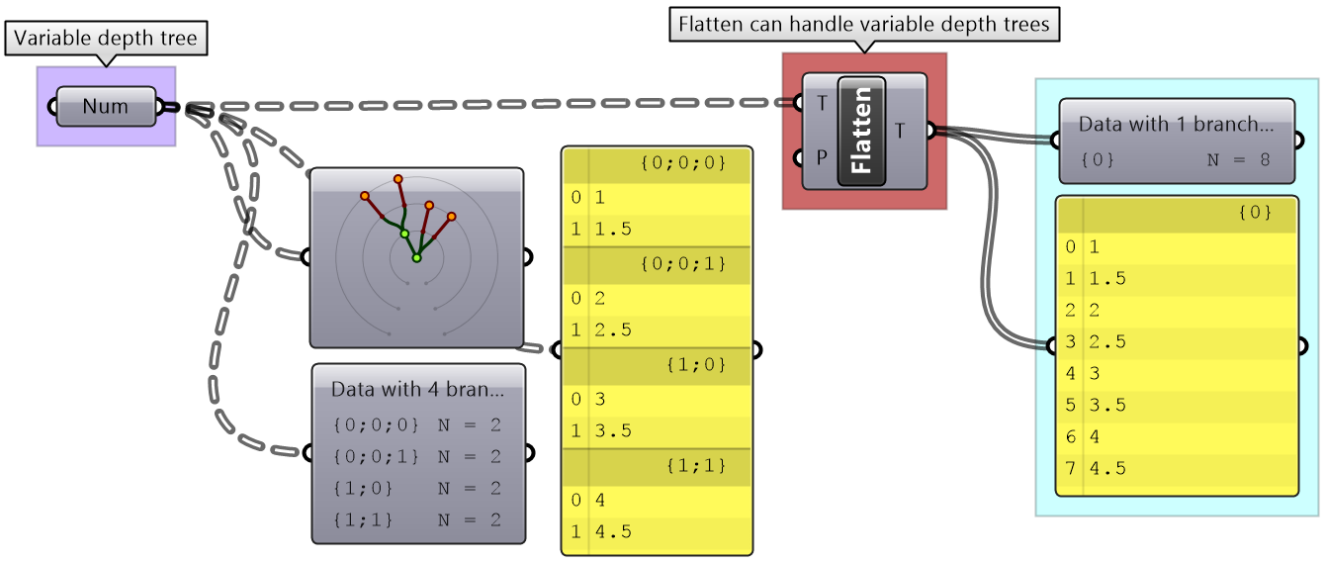


Figura (70): Aplanamiento de árboles complejos

3_5_5: Combinación de flujos de datos

Es posible crear una serie de listas en un árbol donde cada lista se convierte en una rama de un árbol nuevo. Es diferente de la fusión de listas en la que simplemente se crea una lista más grande.

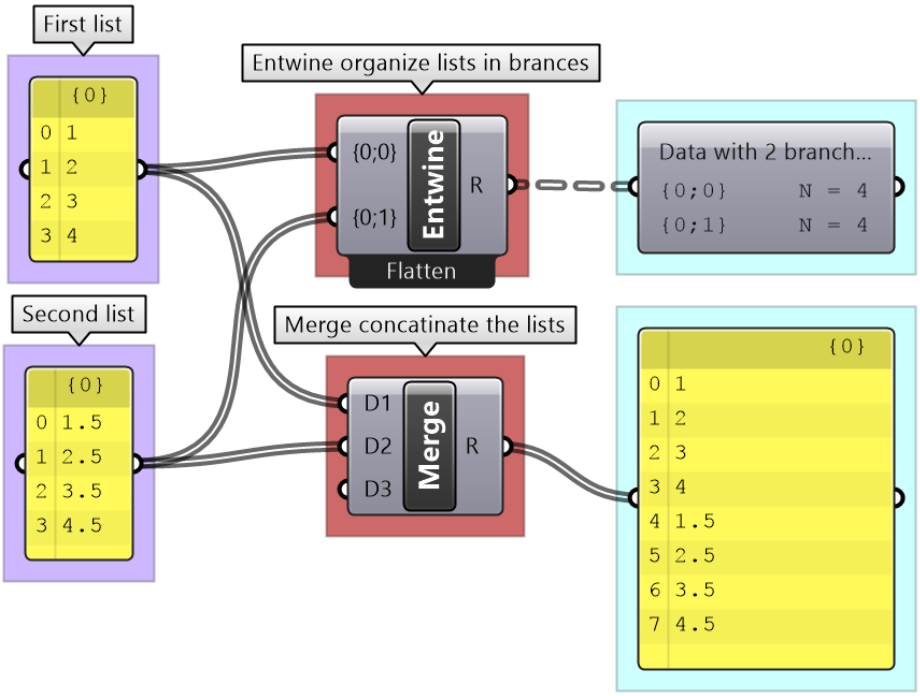


Figura (71): Los componentes ***Entwine*** y ***Merge*** combinan listas en árboles o listas más grandes

3_5_6: Invertir la estructura de datos

En algunos casos es lógico invertir el árbol para cambiar la dirección de las ramas, lo cual es especialmente útil en las rejillas cuando los puntos se organizan en filas y columnas (similar a una estructura de matriz bidimensional). La inversión hace que los elementos correspondientes de las ramas (que tienen el mismo índice en su rama) se agrupen en una rama. Por ejemplo, un árbol de datos que tiene 2 ramas y 4 elementos en cada rama, puede convertirse en un árbol con 4 ramas y 2 elementos en cada rama.

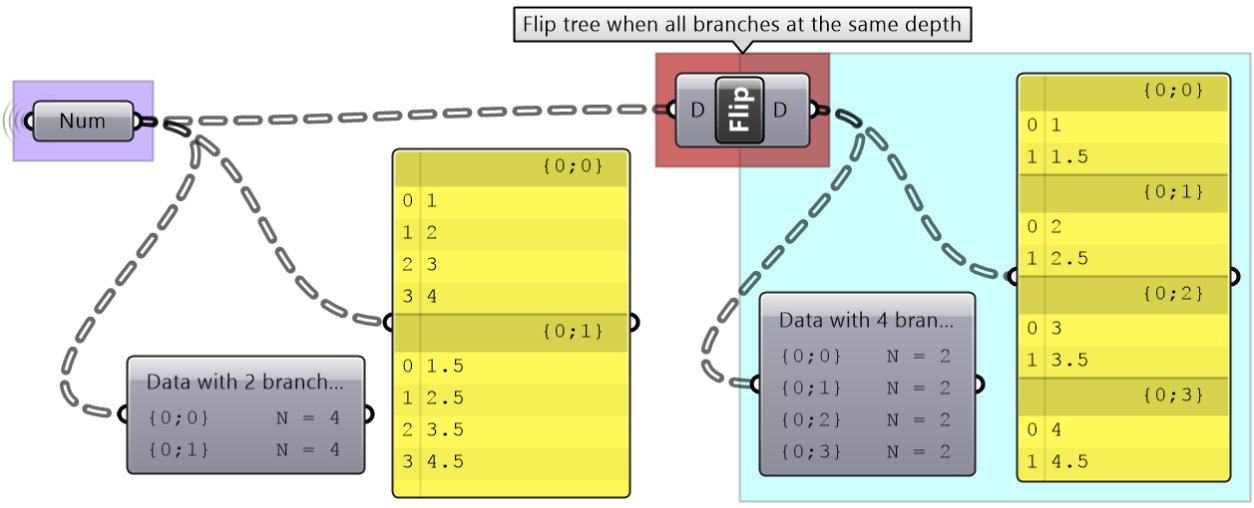


Figura (72): El componente ***Flip*** ayuda a reorganizar los árboles de datos

Si el número de elementos de las ramas es de longitud variable, algunas de las ramas del árbol invertido tendrán valores "null".

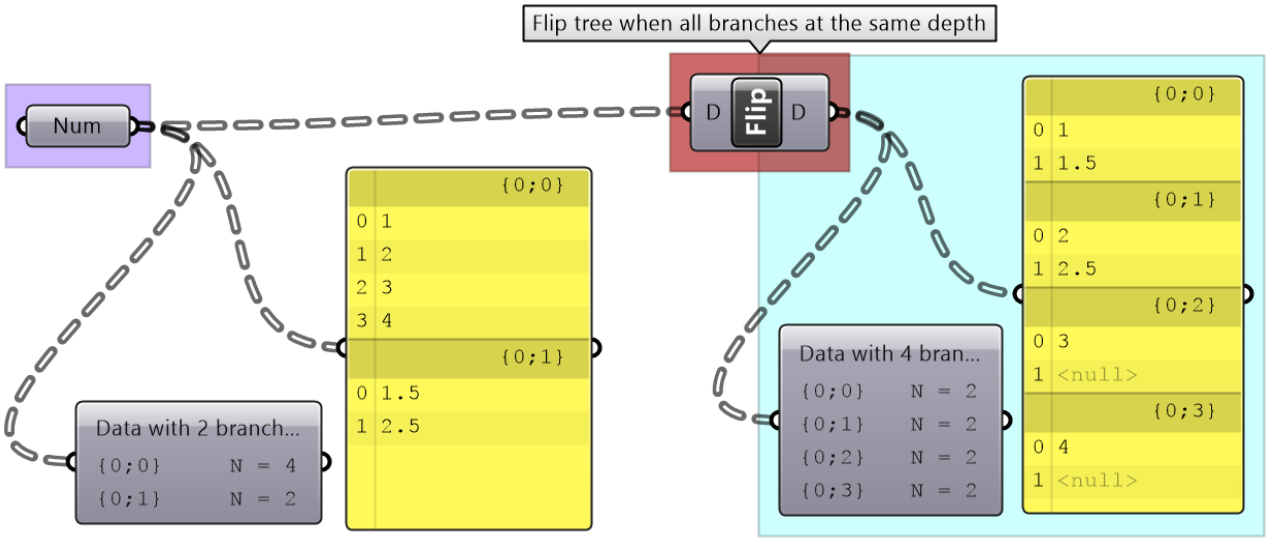


Figura (73): Se añade "null" al invertir árboles con ramas de longitud variable

La inversión es una de las operaciones que no puede utilizar ramas de profundidad variable, simplemente porque no hay una solución lógica para la inversión.

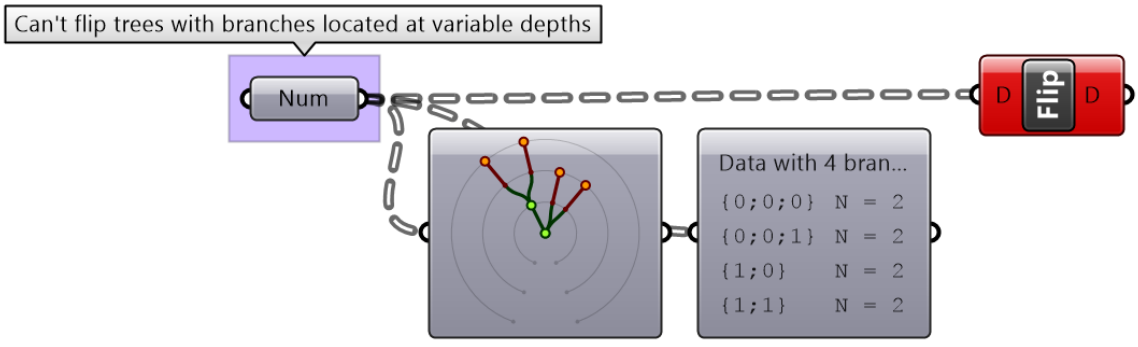


Figura (74): El componente ***Flip*** falla cuando el árbol de entrada tiene ramas de profundidad variable

3_5_7: Simplificar la estructura de datos

El procesamiento de datos a través de múltiples componentes puede añadir una complejidad innecesaria a la estructura de datos. La forma más común es añadir ceros iniciales o finales a las direcciones de las rutas. Las estructuras de datos complejas son difíciles de igualar. ***Simplify Tree*** ayuda a eliminar las ramas vacías. Existen otras operaciones como ***Clean Tree*** y ***Trim Tree*** que ayudan a eliminar elementos "null", ramas vacías y a reducir la complejidad. También es posible extraer todas las ramas como listas independientes utilizando la operación ***Explode Tree***.

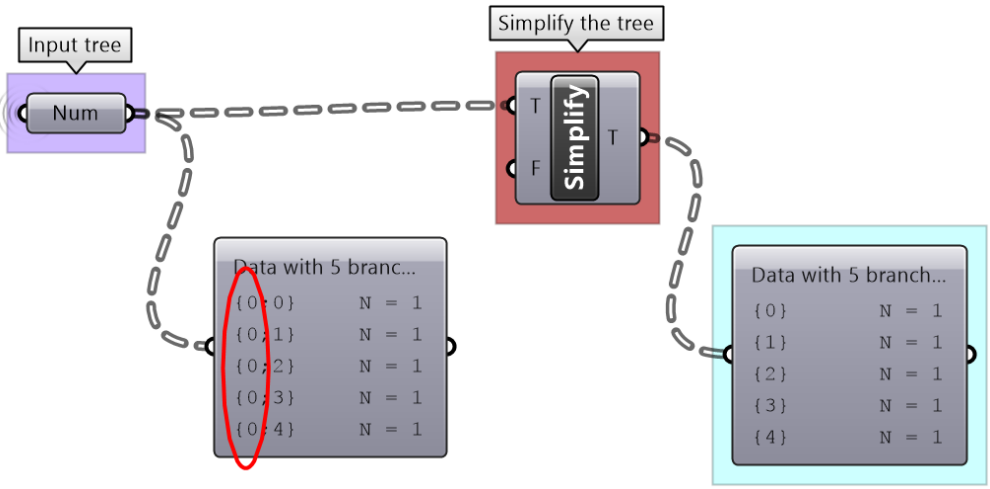
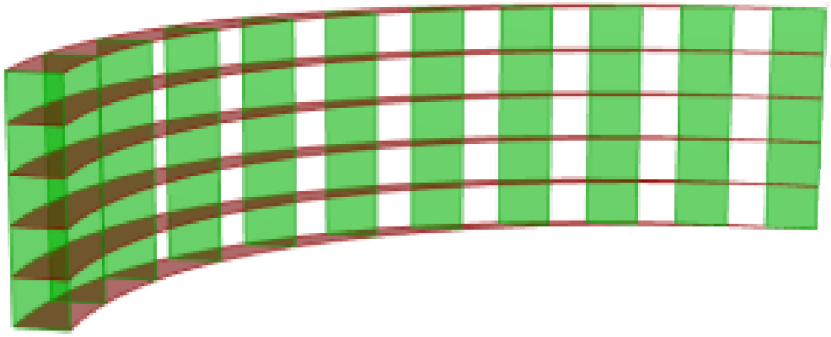
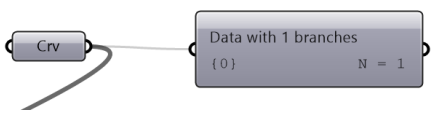
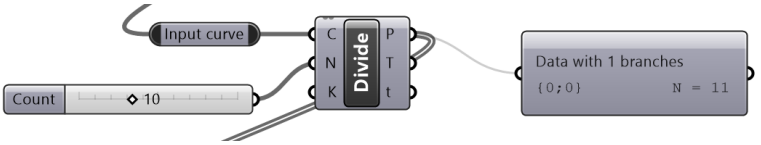


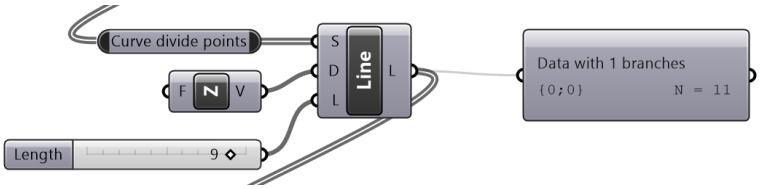
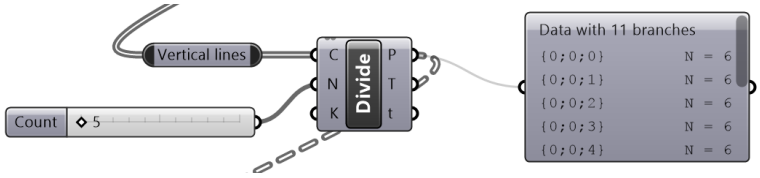
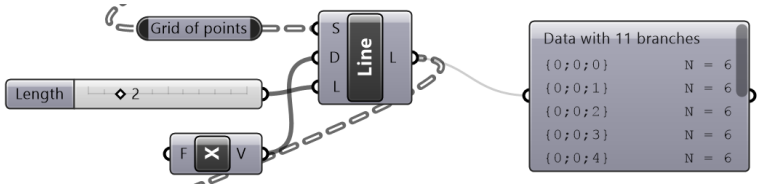
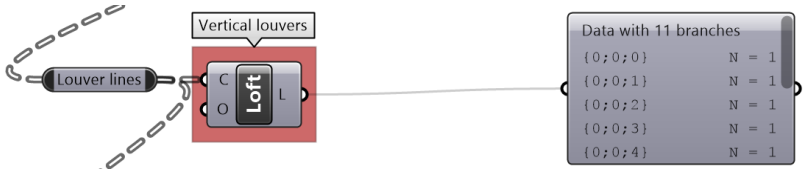
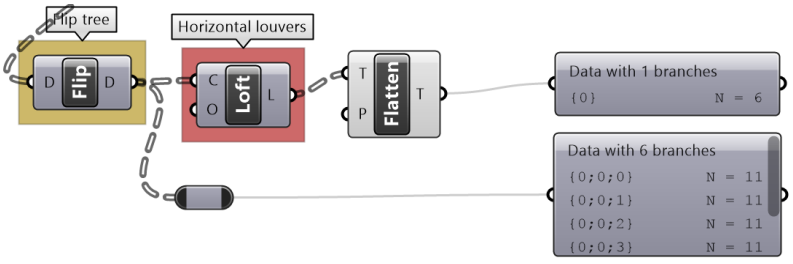
Figura (75): Las rutas pueden aumentar su complejidad a medida que se aplican más operaciones a los datos. ***Simplify*** ayuda a eliminar las ramas vacías.

3_5_8 Tutorial de operaciones básicas con árboles 1

Dada una curva en el plano XY, cree rejillas horizontales y verticales como en la imagen:

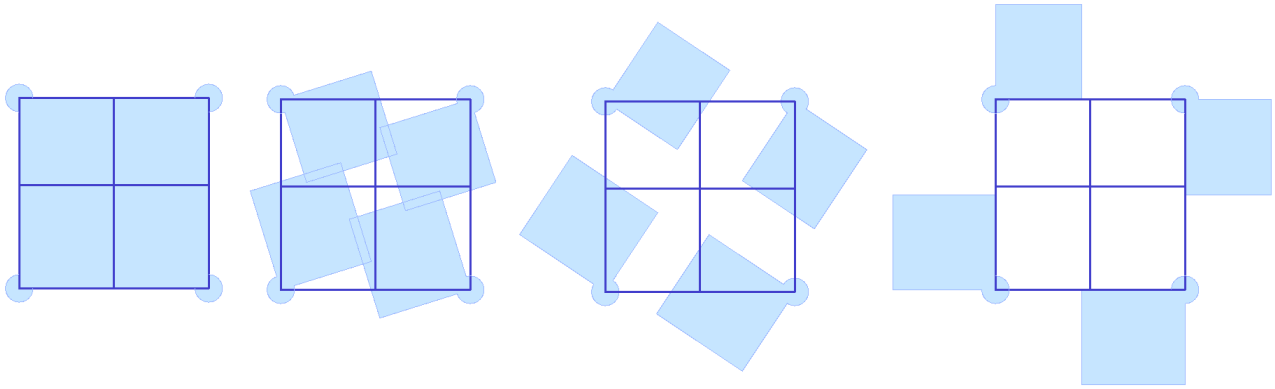


Solución	
Curva de entrada Estructura de datos: elemento único (una rama y un elemento en la rama)	
Dividir la curva para extraer **puntos**. Estructura de datos: lista (una rama con 11 elementos). Observe que la ruta ha añadido un "0" inicial". Esto indica el siguiente nivel de cálculo.	

Cree **líneas** verticales en cada punto. Estructura de datos: lista (una rama con 11 elementos). Observe que la ruta no ha aumentado en complejidad.	
Divida las líneas verticales para crear una **rejilla de puntos**. Estructura de datos: Árbol (11 ramas con 6 elementos). Observe que la ruta ha añadido un "0" inicial".	
Cree **líneas** horizontales en cada punto. Estructura de datos: Árbol (11 ramas con 6 elementos). Observe que la ruta no ha aumentado en complejidad.	
Cree **superficies** de transición a través de las ramas de líneas. Estructura de datos: Árbol (11 ramas con 1 elemento cada una). Observe que la ruta no ha aumentado en complejidad.	
Invierta la matriz del árbol y luego cree **superficies** de transición a través de las ramas de líneas. Estructura de datos: Árbol (11 ramas con 1 elemento cada una). Observe que la ruta no ha aumentado en complejidad. Puede **aplanar** el árbol para crear una lista de rejillas horizontales.	

3_5_9 Tutorial de operaciones básicas con árboles 2

Dados cuatro puntos de esquina en un plano y un radio para la bisagra, cree una persiana que pueda abrirse y cerrarse como en la imagen utilizando un parámetro de rotación.

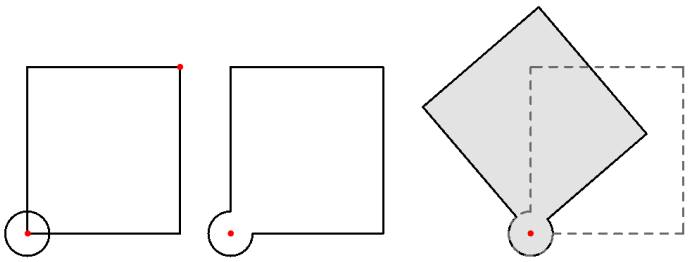


Análisis del algoritmo

Para cada persiana hay dos partes: el rectángulo y la bisagra.

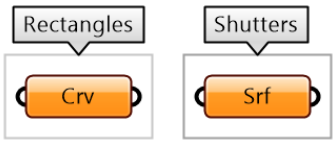
Una el rectángulo y la bisagra, y luego permita la rotación alrededor de la bisagra.

Hay un control de rotación para mover todas las persianas juntas.

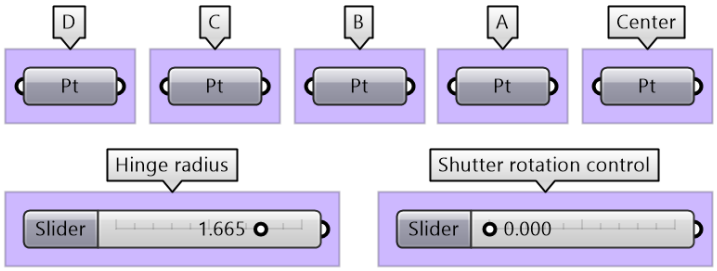


Implementación en Grasshopper

Salida
Superficie de las persianas
Curvas para el marco

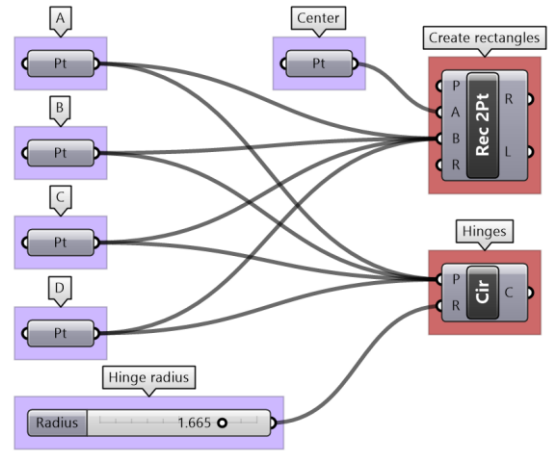


Entrada
4 puntos de esquina (y centro)
Radio de la bisagra
Parámetro de rotación

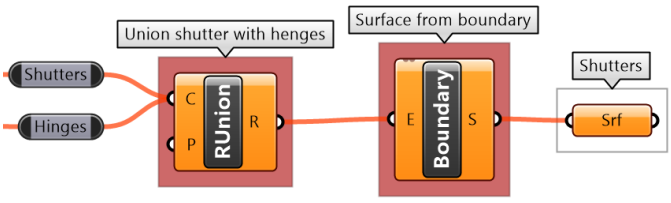


Procesos clave

Cree rectángulos y bisagras. Utilice el componente ***Rectangle***

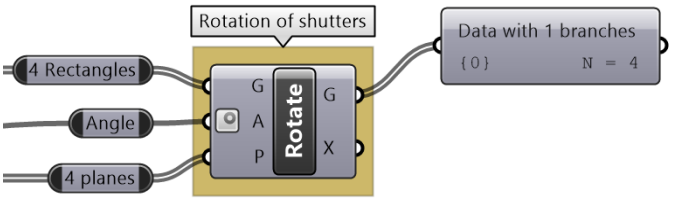


Una las curvas. Utilice el componente ***RUnion***
Cree una superficie a partir del contorno. Utilice el componente ***Boundary***.



Proceso intermedio 1

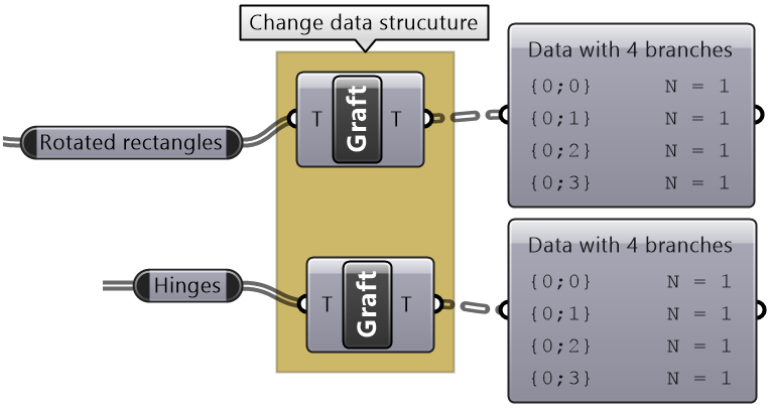
Gire los rectángulos utilizando el ángulo. Utilice el componente ***Rotate***.



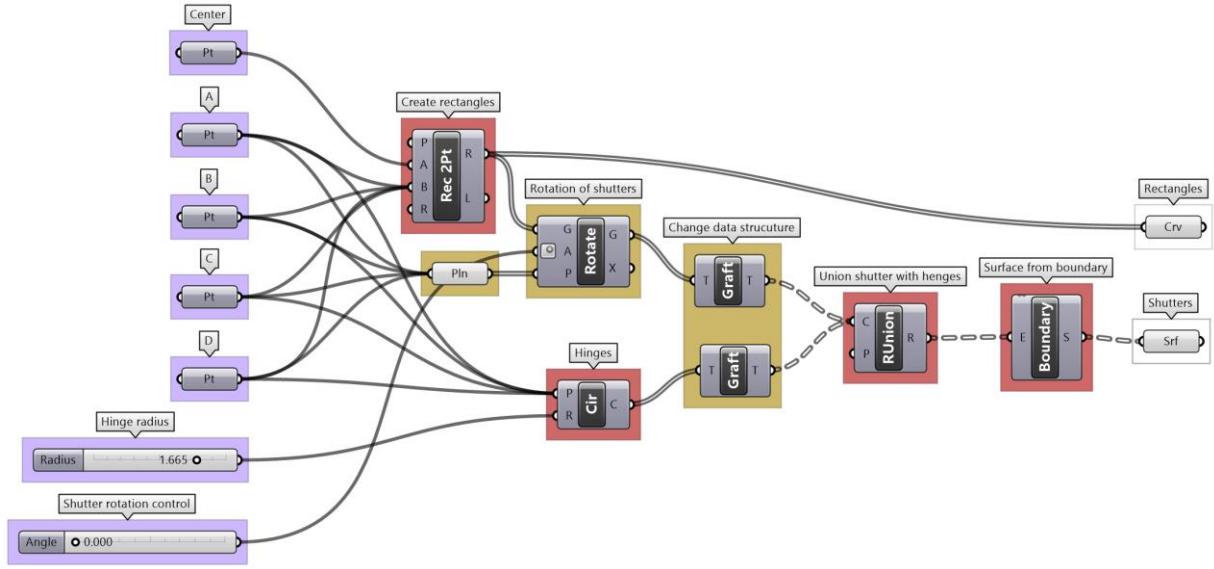
Proceso intermedio 2

Iguale correctamente las estructuras de datos de los rectángulos y de las bisagras antes de la unión de regiones.

Utilice el componente ***Graft*** para que los rectángulos y las bisagras se emparejen correctamente.



Junte todos los procesos



3_6: Operaciones avanzadas con árboles

A medida que sus soluciones sean más complejas, también lo serán las estructuras de datos. Explicaremos tres operaciones avanzadas con árboles que son necesarias para resolver problemas específicos o que se utilizan para simplificar la solución aprovechando directamente las capacidades de la estructura del árbol de datos.

3_6_1: Elementos relativos

La primera operación tiene que ver con la resolución del problema general de la conectividad entre los elementos de un árbol o de varios árboles. Supongamos que tiene una rejilla de puntos y quiere

conectar los puntos en diagonal. Para cada punto, conecte con otro punto en la rama +1 y el índice +1. Por ejemplo, un punto de la rama {0}, índice [0], se conecta con el punto de la rama {1}, índice [1].

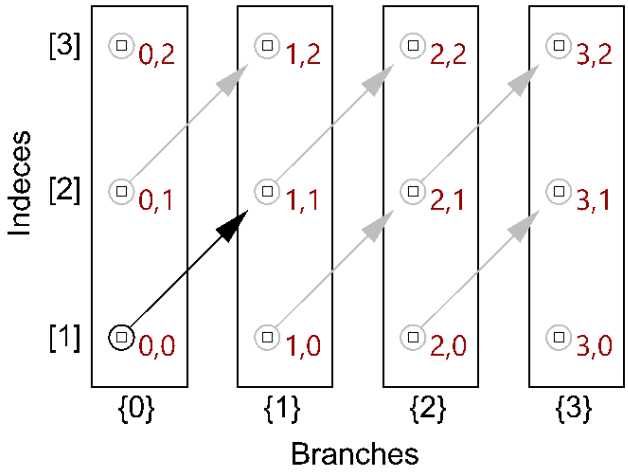


Figura (76): La máscara de elemento relativo {+1}[+1] crea una conectividad diagonal positiva

En Grasshopper, la forma de comunicar el desfase se expresa con una cadena de desfase con el formato "{desfase de rama}[desfase de índice]". En nuestro ejemplo, la cadena para conectar puntos en diagonal es "{+1}[+1]". Este es un ejemplo que utiliza el componente de árbol relativo en Grasshopper. Observe que el componente de elemento relativo crea dos árboles nuevos que se correlacionan de la manera especificada en la cadena de desfase.

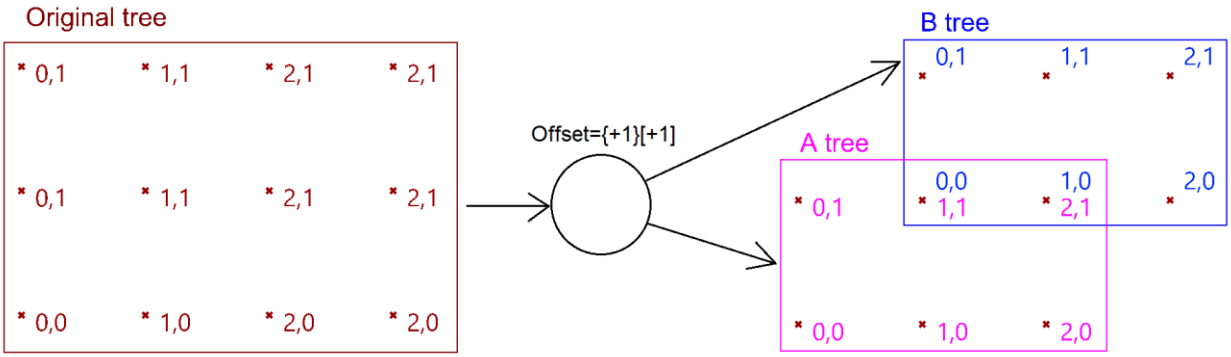


Figura (77): La máscara de elemento relativo {+1}[+1] rompe el árbol original en 2 nuevos árboles con conectividad diagonal

Este es un ejemplo de implementación en Grasshopper en el que definimos elementos relativos en un árbol y luego conectamos los dos árboles resultantes con líneas utilizando el componente ***Relative Item***.

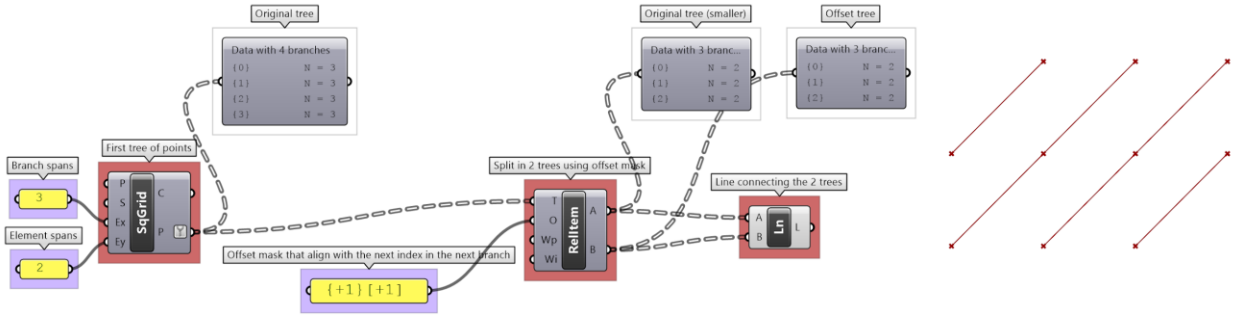
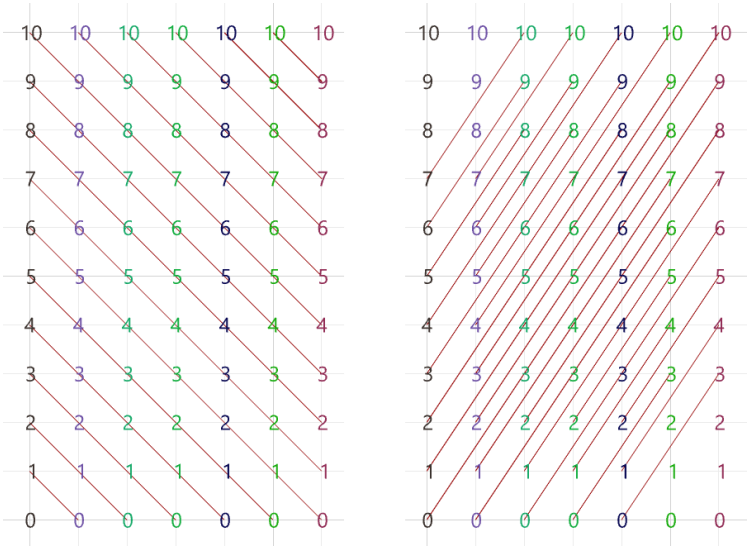
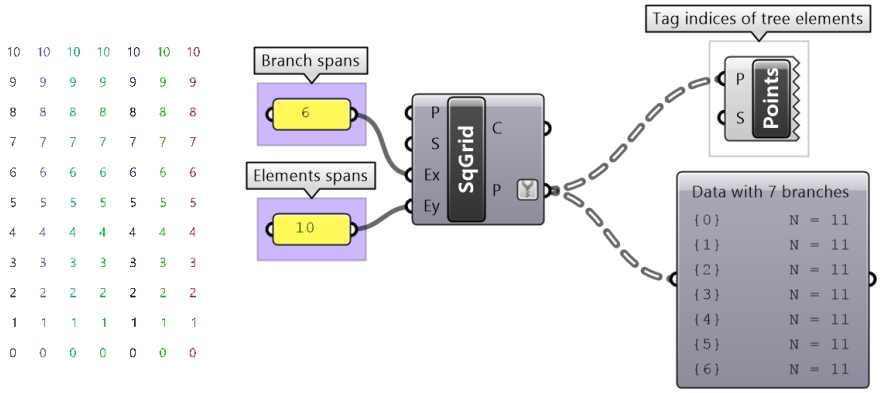
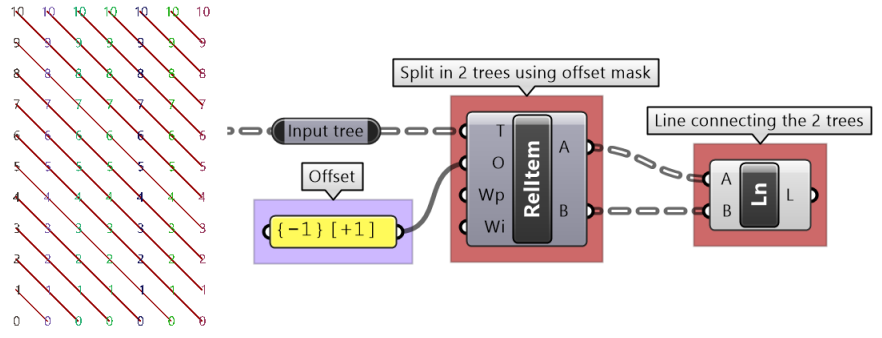


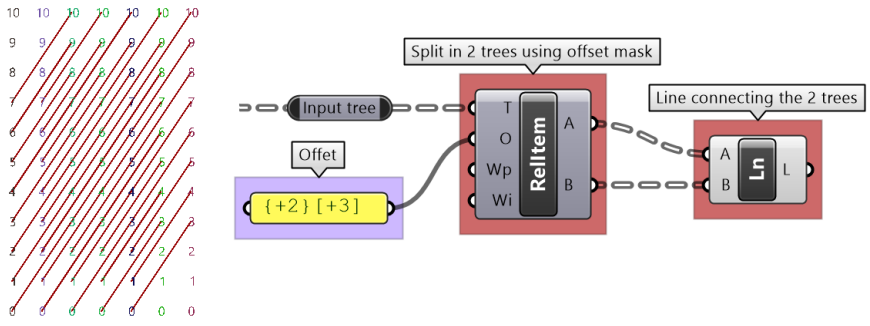
Figura (78): Componente ***Relative Item*** con máscara {+1}[+1] en Grasshopper

3_6_1_1 Tutorial de elementos relativos 1

Cree el patrón que se muestra en la imagen utilizando una rejilla de 7 ramas donde cada rama tiene11 elementos.



Solución	
Define el {desfase de rama} [desfase_de_índice]	
Cree la rejilla.	
Cree árboles relativos que conecten cada elemento con la rama -1 y el índice: {-1}[+1] Crea líneas para conectar los 2 árboles relativos.	

Cambia el desfase a {+2}[+3] para crear las segundas conexiones.	

Hemos mostrado cómo definir elementos relativos en un árbol, pero también se pueden especificar elementos relativos entre 2 árboles. Tendrá que prestar atención a la estructura de datos de los dos árboles de entrada y asegurarse de que son compatibles. Por ejemplo, si conecta cada punto del primer árbol con otro punto de otro árbol con el mismo índice, pero con más de una rama, entonces puede definir la cadena de desfase en {+1}[0].

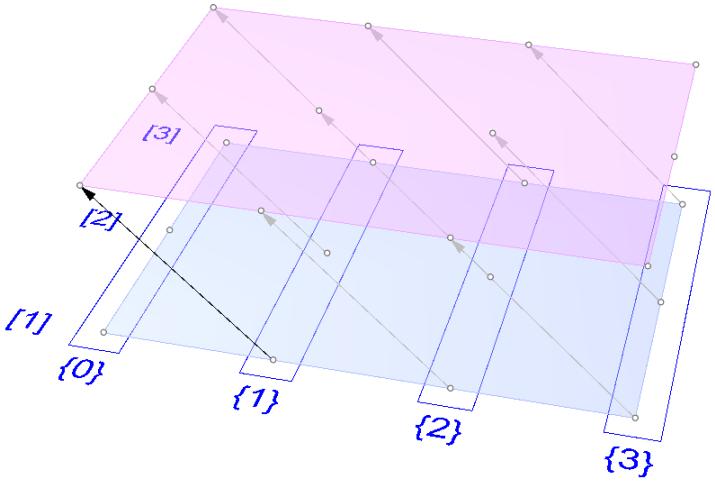


Figura (79): ***Relative Items*** crea conexiones en varios árboles

La entrada al componente ***Relative Items*** son dos árboles y la salida son dos árboles con los elementos correspondientes según la cadena de desfase.

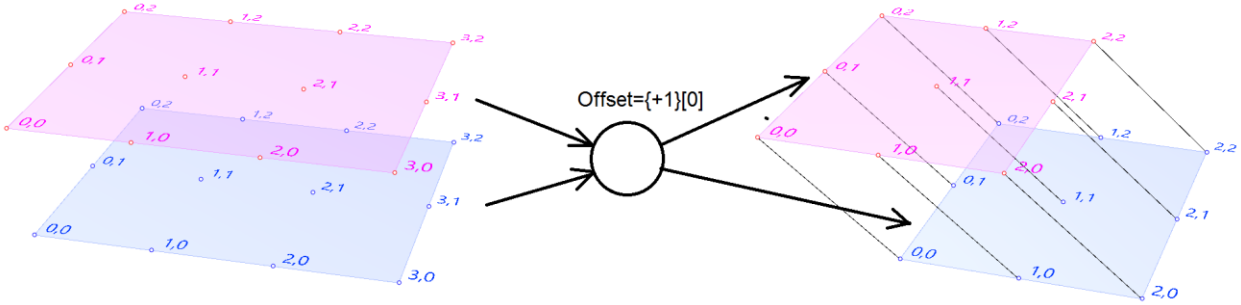


Figura (80): La máscara de desfase de ***Relative Items*** genera nuevos árboles con las conexiones deseadas.

La siguiente definición de GH refleja lo anterior:

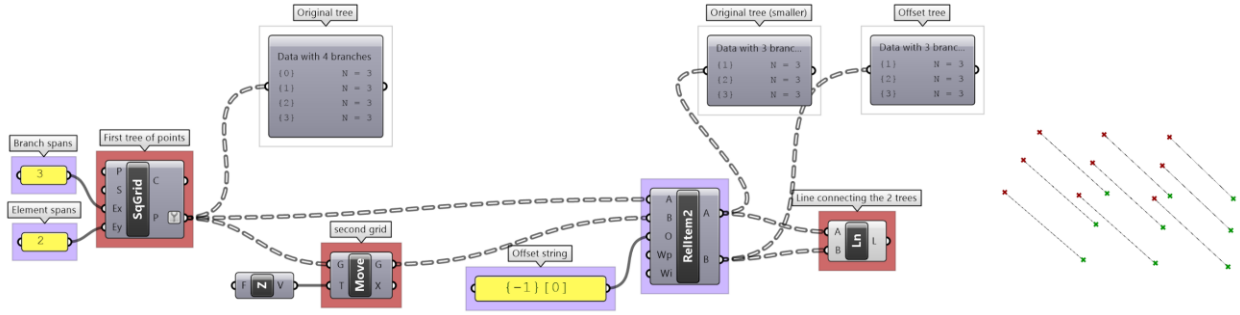
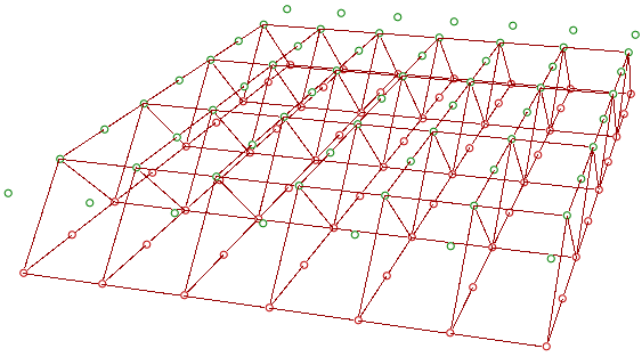
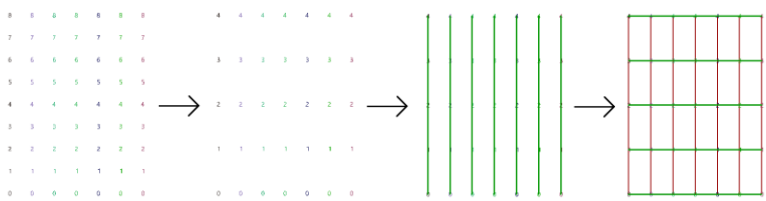
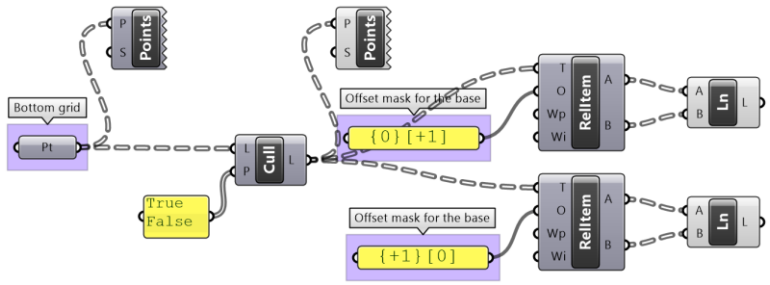


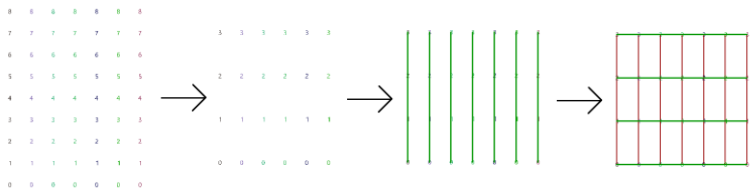
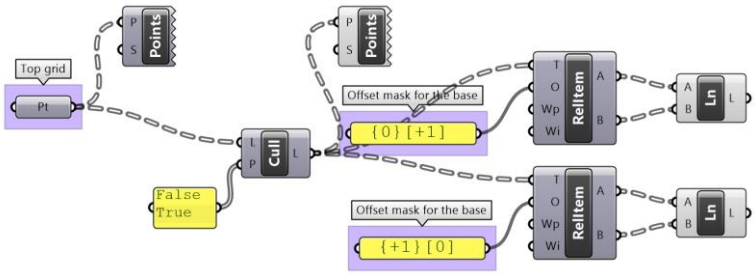
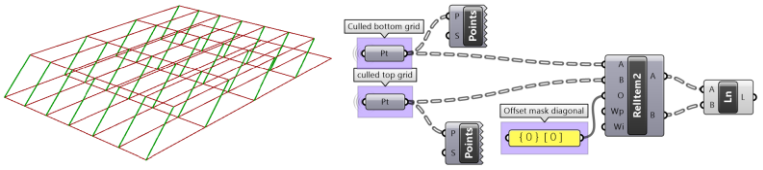
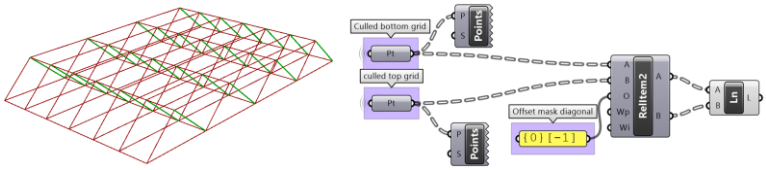
Figura (81): Implementación de ***Relative Items*** en Grasshopper

3_6_1_2 Tutorial de elementos relativos 2

Utilice elementos relativos entre 2 rejillas delimitadoras para generar la estructura que se muestra en la imagen.



Solución	
Conexiones de árbol inferior	
Oculte cada dos índices y mantenga el mismo número de ramas (oculte los índices 1, 3, etc.). Defina las cadenas de desfase de los componentes de *RelativeItem* para crear las conexiones verticales y horizontales.	
Definición de Grasshopper	
Conexiones de árbol superior	

Oculte cada dos índices y mantenga el mismo número de ramas (oculte los índices 0, 2, etc.). Defina las cadenas de desfase de los componentes de *RelativeItem* para crear las conexiones verticales y horizontales.	
Definición de Grasshopper	
Conexiones entre los 2 árboles	
Utilice las rejillas seleccionadas y, a continuación, defina la primera cadena de desfase del componente *RelativeItems* para crear el primer conjunto de líneas transversales: {0}[0]	
Defina la segunda cadena de desfase del componente *RelativeItems* para definir el segundo conjunto de líneas transversales: {0}[-1]	

3_6_2: Dividir árboles

La capacidad de seleccionar una parte de un árbol o de dividirlo en dos partes es una operación de árbol muy potente en Grasshopper. Puede dividir el árbol utilizando una máscara de cadena que especifica la salida positiva del árbol, y lo que queda se denomina "árbol negativo" y se usa como salida. Puesto que todos los árboles tienen ramas e índices, la máscara de división debe incluir información sobre qué ramas y qué índices de esas ramas hay que dividir. Estas son las reglas de la máscara de división:

Máscara de división de árbol: sintaxis y reglas generales	
{ ; ; }	Utilice corchetes para incluir la máscara de las ramas del árbol.
[ ]	Utilice paréntesis cuadrados para incluir la máscara de los elementos (hojas). Puede omitirse si se seleccionan todos los elementos o se utiliza [*].
( )	Los paréntesis se utilizan para organizar y agrupar.
*	Cualquier número de enteros en una ruta. El asterisco también permite incluir todas

	las ramas, sin importar cómo son las rutas.
?	Cualquier número entero
6	Cualquier número entero específico
!6	Cualquier elemento excepto un número entero específico
(2,6,7)	Cualquiera de los enteros específicos de este grupo.
!(2,6,7)	Cualquier elemento excepto uno de los enteros de este grupo.
(2 a 20)	Cualquier número entero en este rango (incluidos 2 y 20).
!(2 a 20)	Cualquier número entero fuera de este rango.
(0,2,...)	Cualquier número entero que forme parte de esta secuencia infinita. Las secuencias deben tener al menos dos enteros, y cada entero subsiguiente tiene que ser mayor que el anterior (puede que sea una limitación temporal).
(0,2,...,48)	Cualquier entero que forme parte de esta secuencia finita. Tiene la opción de introducir un límite de secuencia después de los tres puntos.
!(3,5,...)	Cualquier número entero que no forme parte de esta secuencia infinita. La secuencia no se extiende hacia la izquierda, solo hacia la derecha. Esta regla seleccionaría los números 0, 1, 2, 4, 6, 8, 10, 12 y todos los números pares restantes.
!(7,10,21,...,425)	Cualquier número entero que no que forme parte de esta secuencia finita.
{ * }[(0 a 4) **o** (6,11,41) **]**	Es posible combinar dos o más reglas utilizando los las booleanas y los operadores. El ejemplo selecciona los cinco primeros elementos de cada lista de un árbol y también los elementos 7, 12 y 42.

Estos son algunos ejemplos de máscaras de división válidas.

División por ramas	
{ * }	Selecciona todo (la salida de todo el árbol como positivo, y el árbol negativo estará vacío)
{ *; 2 **}**	Selecciona la tercera rama
{ *; (0,1) **}**	Selecciona las dos primeras ramas finales
{ *; (0, 2, …) **}**	Selecciona todas las ramas pares
División por ramas y hojas	
{ * }[(1,3,...)**]**	Selecciona los elementos situados en los índices impares de todas las ramas

{ *; 0 **}[**(1,3,...)**]**	Selecciona los elementos situados en los índices impares de la primera rama
{ *; (0, 2) **}[**(1,3,...)**]**	Selecciona los elementos situados en los índices impares de la primera y la tercera rama
{*; (0,2,...) **} [** (1,3,...) **]**	Selecciona los elementos situados en los índices impares de las ramas situadas en índices pares
{*; (0,2,...) **} [**(0) **o** (1,3,...)**]**	Selecciona los elementos situados en los índices impares, y el índice "0", en las ramas situadas en los índices pares

Una de las aplicaciones comunes que utiliza la función de división de árbol es cuando tiene una rejilla de puntos de la que quiere transformar un subconjunto. En la división, la estructura del árbol original se mantiene y los elementos que se dividen se sustituyen por "null". Por tanto, al aplicar la transformación al árbol dividido, es fácil volver a recombinarlo.
Suponga que tiene una rejilla con 7 ramas y 11 elementos en cada rama, y que quiere desplazar elementos entre los índices 1-3 y 7-9. Puede utilizar la división de árbol para ayudar a aislar los puntos que tiene que mover utilizando la máscara: {*}[(1,2,3) or (7,8,9)], mover el árbol positivo y luego volver a recombinarlo con el árbol negativo.

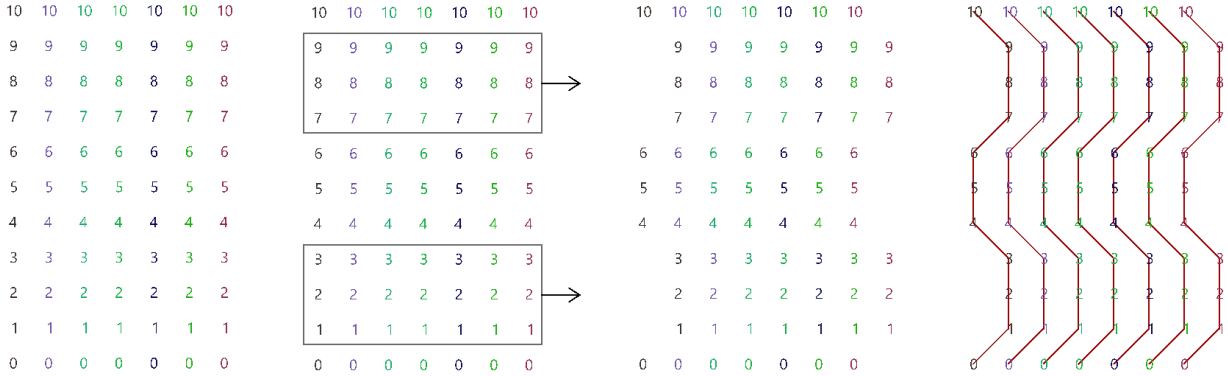


Figura (82): ***Split tree*** permite operar sobre un subconjunto del árbol con la posibilidad de volver a recombinarlo

Esta es la definición de GH que hace lo descrito utilizando el componente ***Split Tree***.

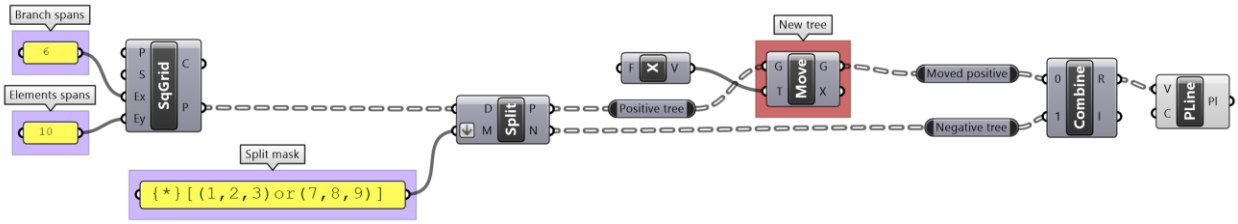
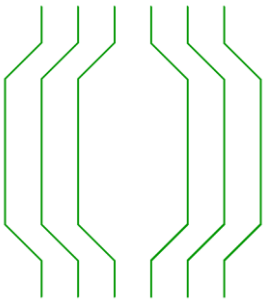


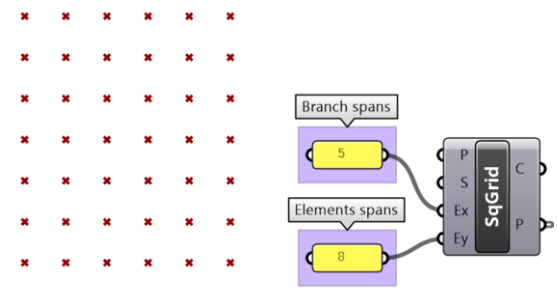
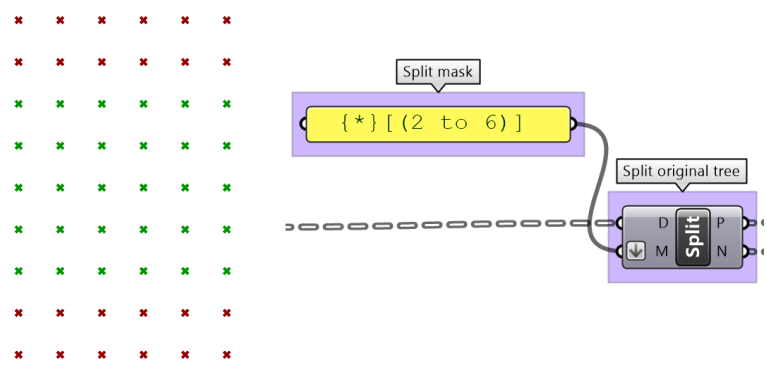
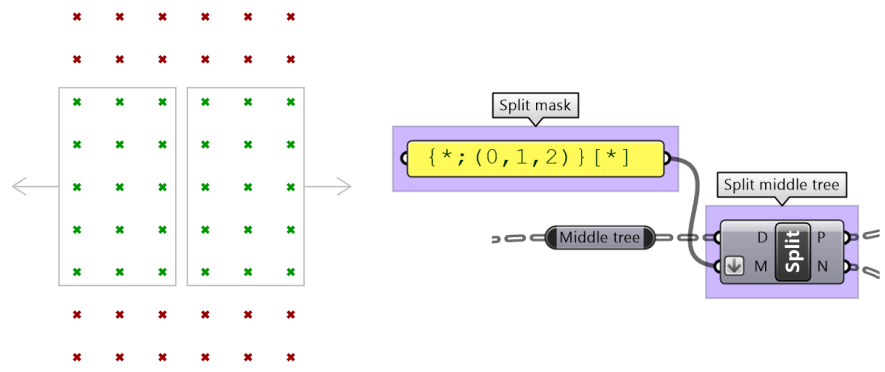
Figura (83): Implementación de ***Split tree*** de Grasshopper de la Ffigura (73)

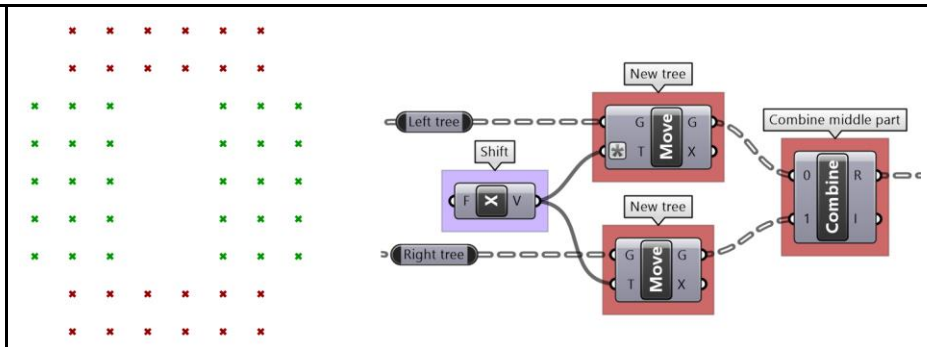
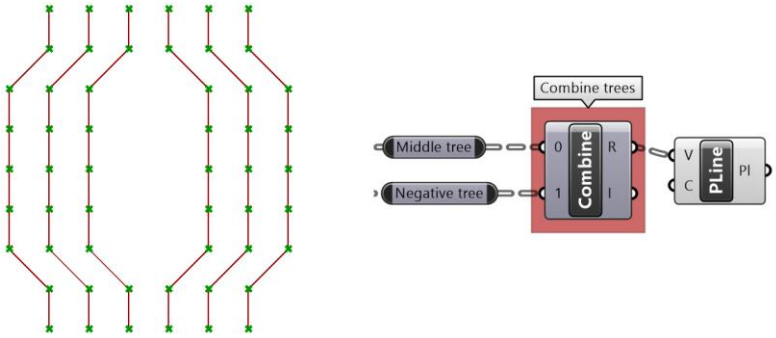
Una de las ventajas de utilizar ***Split Tree*** en árboles relativos es que la máscara de división es muy versátil y es más fácil aislar la parte deseada del árbol. Además, la estructura de datos se mantiene en los árboles negativos y positivos, lo que facilita la recombinación de los elementos del árbol tras el procesamiento de las partes.

3_6_2_1 Tutorial de división de árbol 1

Dada una rejilla de 6x9, utilice la división de árbol para generar la siguiente forma.

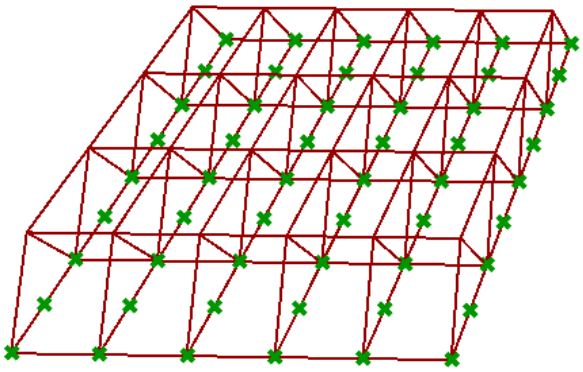


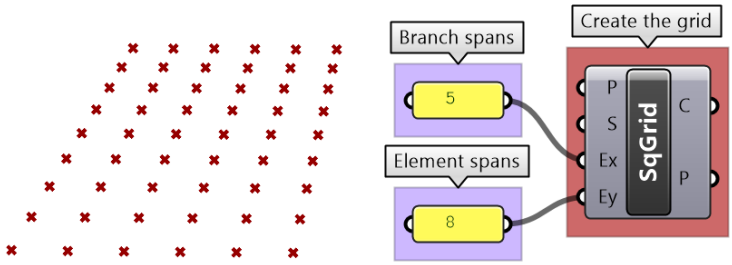
Pasos de la solución	
Cree la rejilla.	
Divida el árbol para aislar la parte central.	
Divida la parte central en dos partes nuevas.	

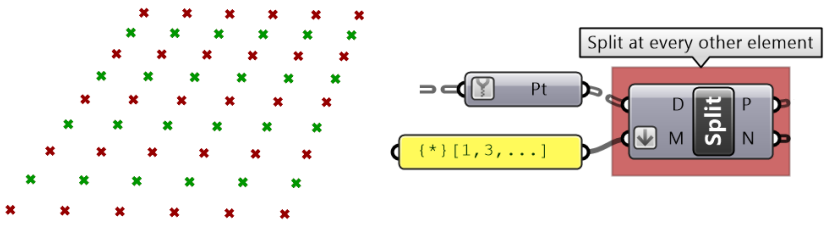
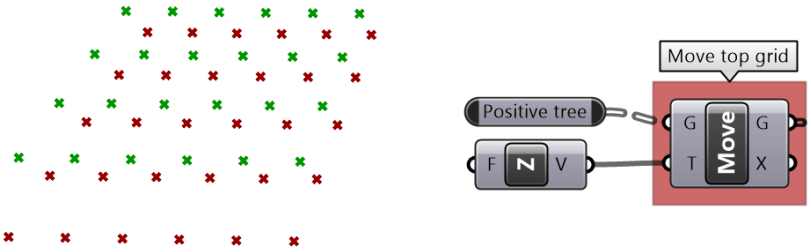
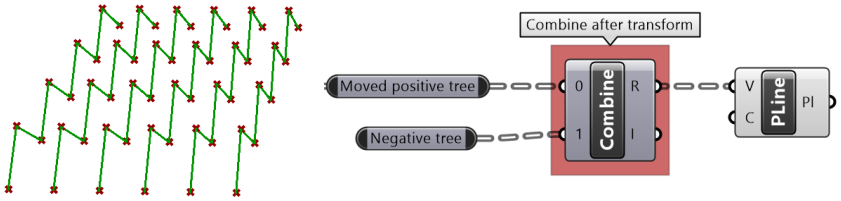
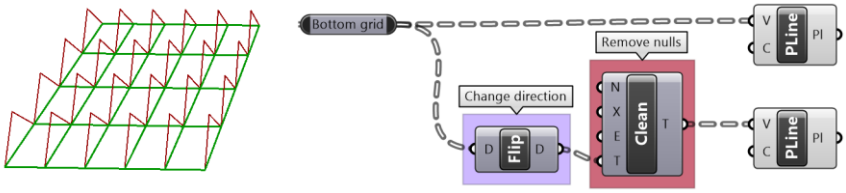
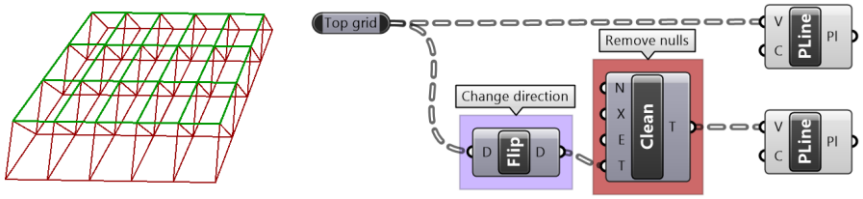
Mueva las dos partes centrales en direcciones opuestas y luego recombínelas.	
Recombine la parte central con el resto del árbol y cree polilíneas a través de los elementos de cada rama.	

3_6_2_2 Tutorial de división de árbol 2

Dada una rejilla, cree el siguiente sistema de celosía utilizando la función de división de árbol.



Solución	
Cree la rejilla de 6x9.	

Divida cada dos elementos.	
Mueva el árbol positivo verticalmente.	
Combine los árboles positivos y negativos. Y cree una polilínea a través de cada elemento de la rama.	
Cree curvas en la parte inferior utilizando el árbol negativo.	
Cree curvas en la parte superior utilizando el árbol positivo.	

3_6_3: Mapeador de ruta

Al tratar con estructuras de datos complejas, como los árboles de datos de Grasshopper, se encontrará con la necesidad de simplificar o reorganizar sus elementos dentro del árbol. Hay algunos componentes que ofrece Grasshopper para ese fin, como ***Flatten***, ***Graft*** o ***Flip***. Aunque son muy útiles, pueden no ser suficientes cuando se trabaja con varios árboles o se necesita una reorganización personalizada. ***Path Mapper*** *es un potente componente de Grasshopper que ayuda a reorganizar los elementos de los árboles o a cambiar la estructura del árbol.* Es quizás el componente menos intuitivo de usar y puede causar pérdidas de datos, pero también es la única manera de encontrar una solución en algunos casos y, por tanto, vale la pena abordarlo aquí.

El ***Path Mapper*** mapea los datos entre las rutas de origen y de destino. La ruta de origen es fija y viene dada por el árbol de entrada. Solo se puede definir la ruta de destino. Hay un conjunto de constantes que ayudan con el mapeado. A continuación se muestra una lista.

item_count	Número de elementos en la rama actual.
path_count	Número de rutas (ramas) del árbol.
path_index	Índice de la ruta actual.

Empecemos familiarizándonos con la sintaxis utilizando los mapeados integrados en ***Path Mappers***.

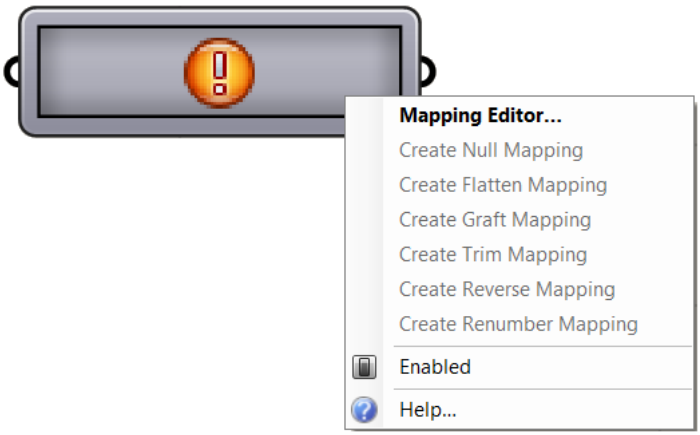
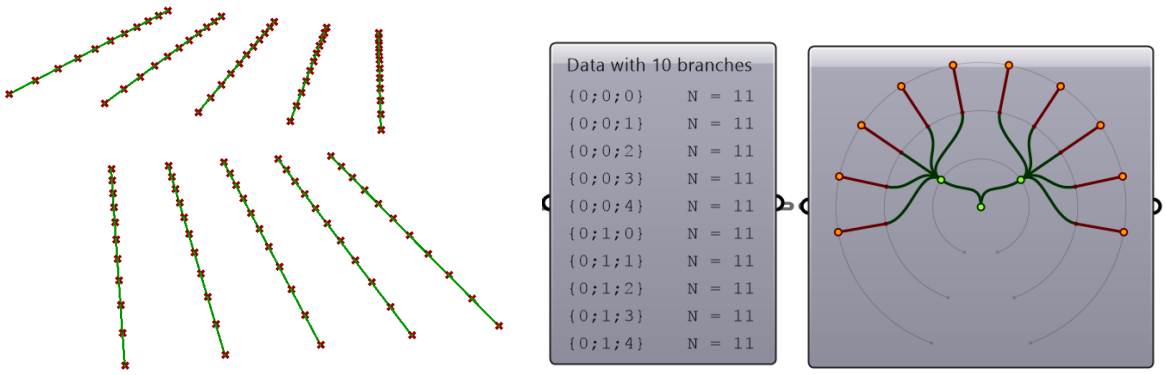
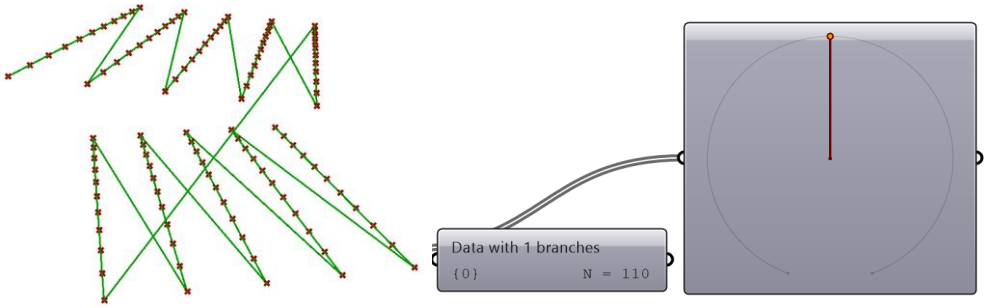
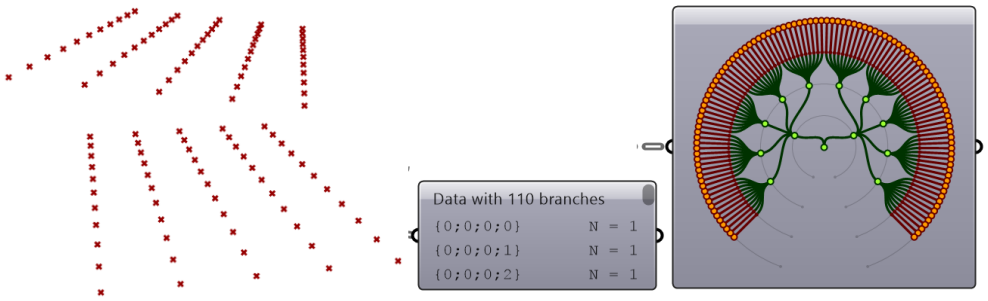
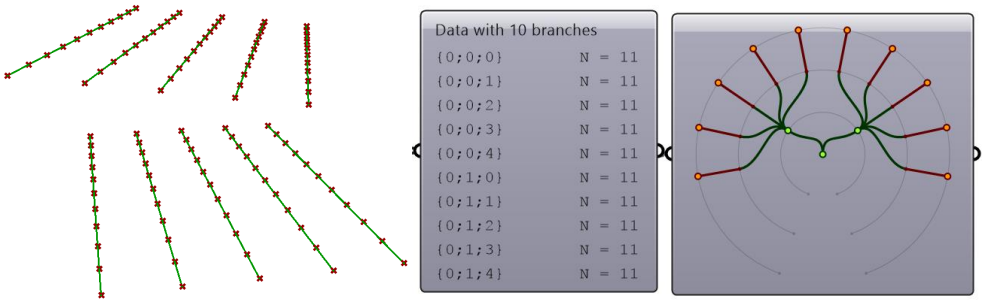
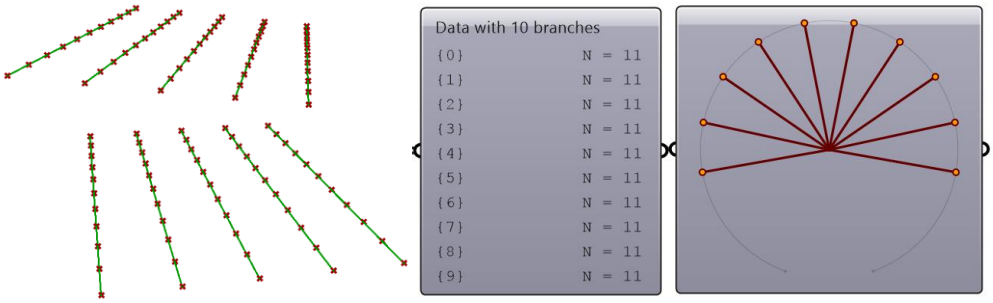


Figura (84): Mapeados integrados en ***Path Mapper***

En el siguiente ejemplo, el árbol de entrada tiene dos rejillas de puntos (2 árboles). La estructura de datos se hace patente cuando se utiliza ***Polyline***, que crea una polilínea a través de cada rama. Examinaremos el efecto de aplicar del mapa integrado en la estructura y la conexión de los puntos.

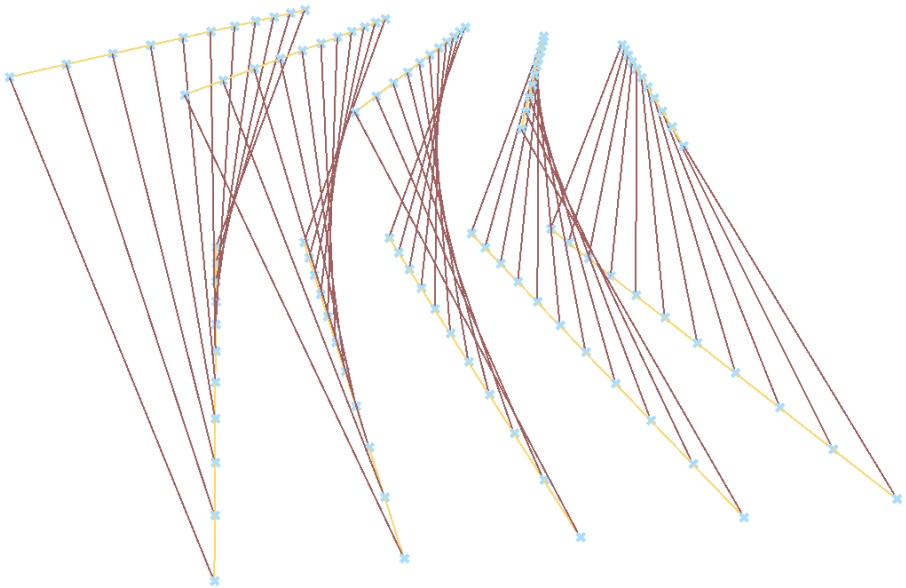


Mapeados integrados en Path Mapper	
Null Mapping	No cambia nada.
Flatten Mapping	

Graft Mapping	
Reverse Mapping	
Renumbering Mapping	

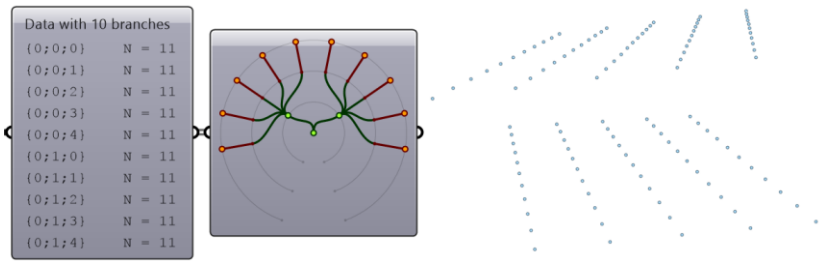
3_6_3_1 Tutorial de mapeado de rutas 1

Dadas las estructuras de árbol de puntos, cree las siguientes conexiones.

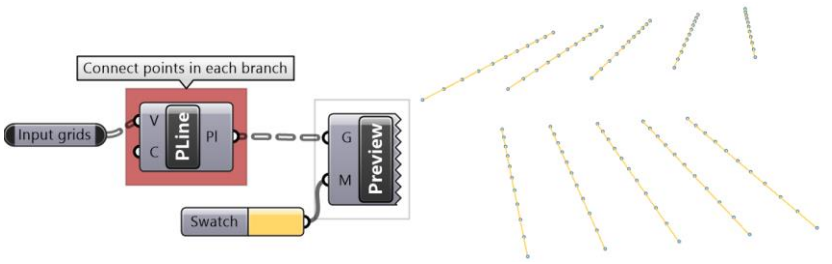


Solución

La entrada tiene dos árboles, y cada uno tiene 5 ramas con 11 elementos en cada rama, un total de 10 ramas.



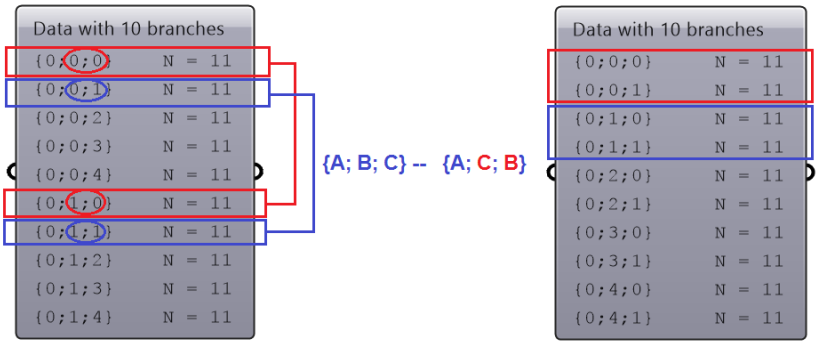
Se puede utilizar ***Polyline*** para conectar los elementos de cada rama.



Para crear las conexiones verticales, es necesario crear una rama para cada 2 elementos correspondientes a través de los 2 árboles y luego utilizar *Polyline* para conectarlos.
1- Analice las rutas de los árboles
2- Realice un mapeado que genere la agrupación deseada

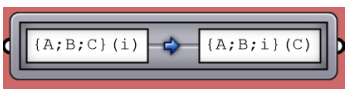
En primer lugar, agrupe las ramas correspondientes en los 2 árboles.

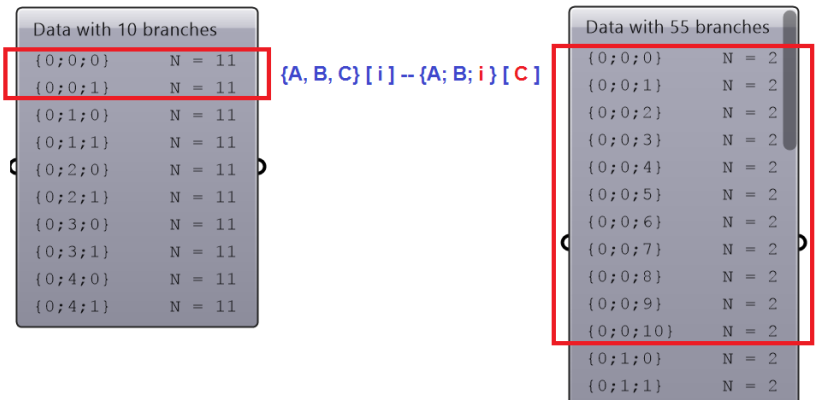
Para hacerlo, cambie los dos últimos números enteros de las rutas:



En segundo lugar, invierta cada uno de los 5 árboles. Como las ramas tienen 11 elementos cada una, al invertir cada árbol se crearán 11 ramas con 2 elementos en cada rama. Un total de 55 ramas.

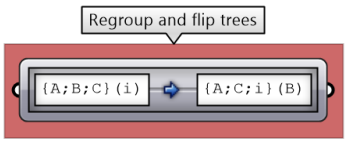
Se invierte cambiando el último entero de la ruta con el índice del elemento:



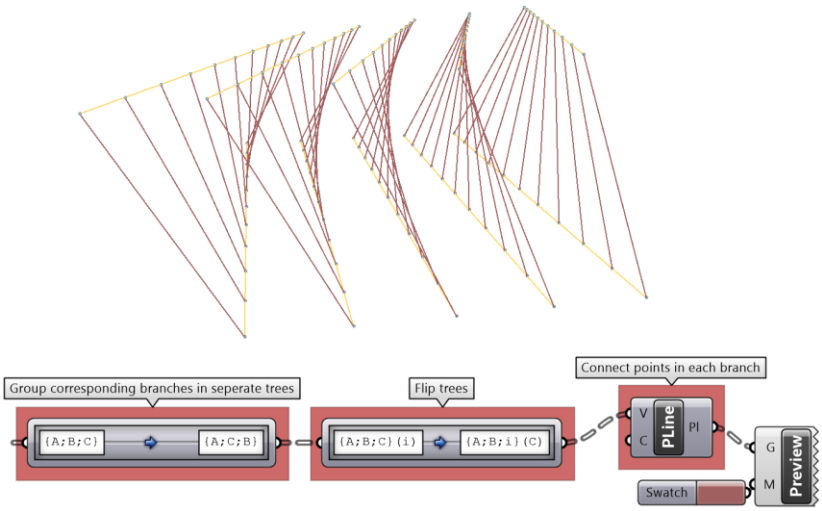


Por último, el componente *Polyline* realiza las conexiones verticales.

Nota: Puede combinar los 2 mapeados en un solo paso como se muestra aquí:

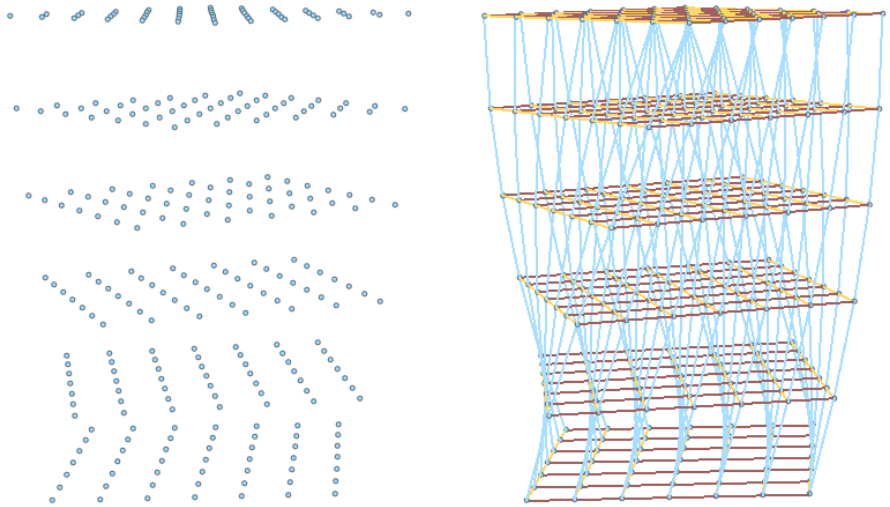


La combinación no siempre es posible, pero puede ahorrar tiempo de procesamiento y reducir el tamaño.



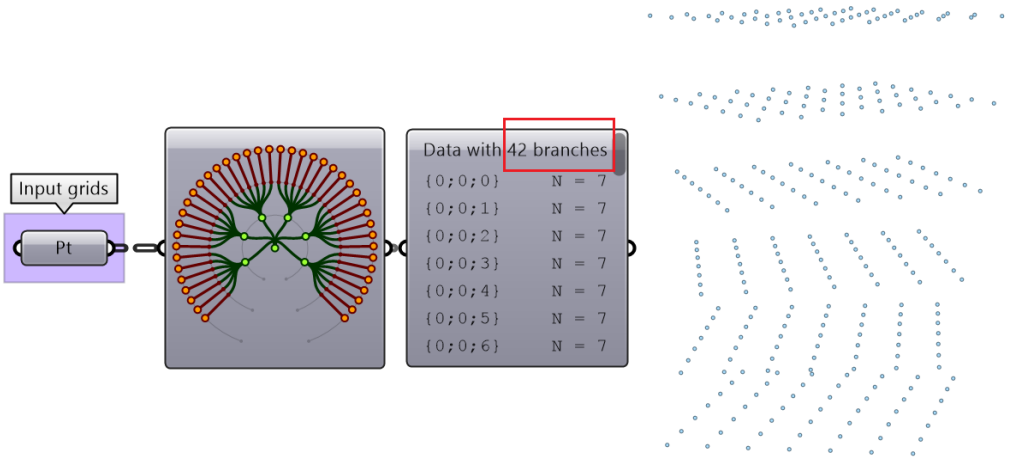
3_6_3_2 Tutorial de mapeado de rutas 2

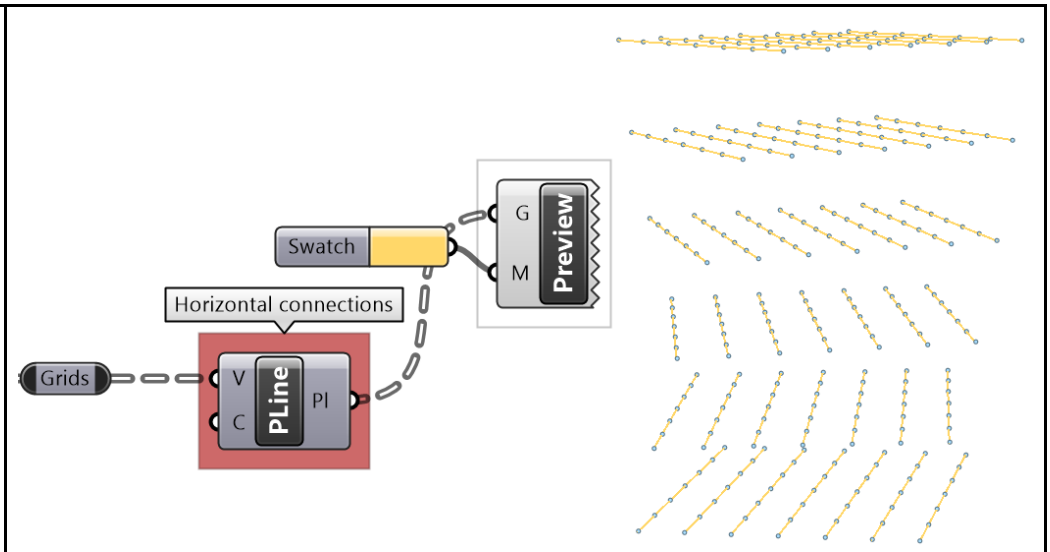
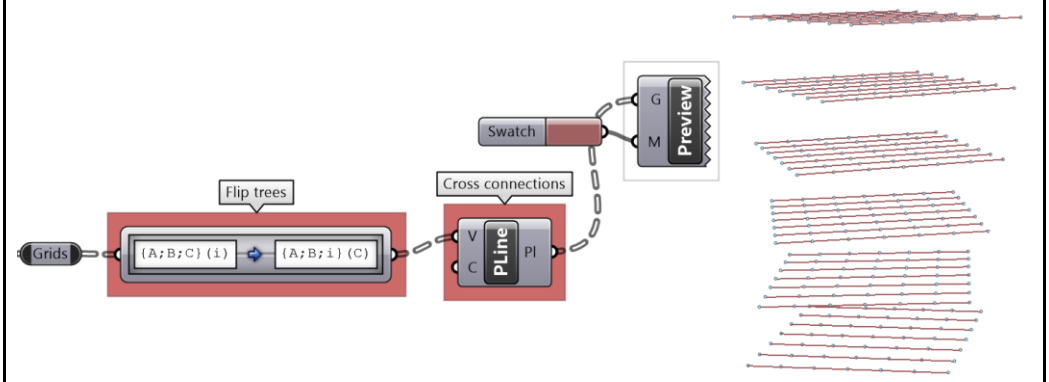
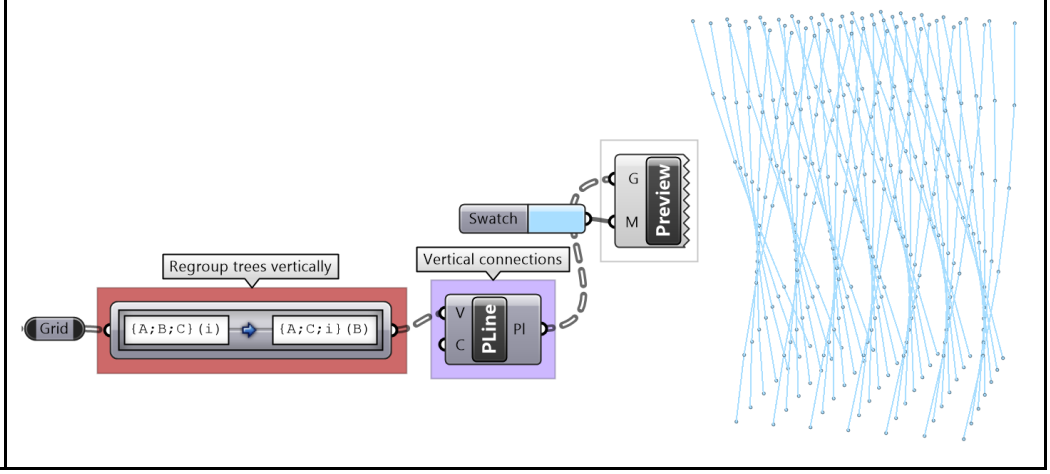
Dado el árbol de puntos de entrada, cree la siguiente estructura.

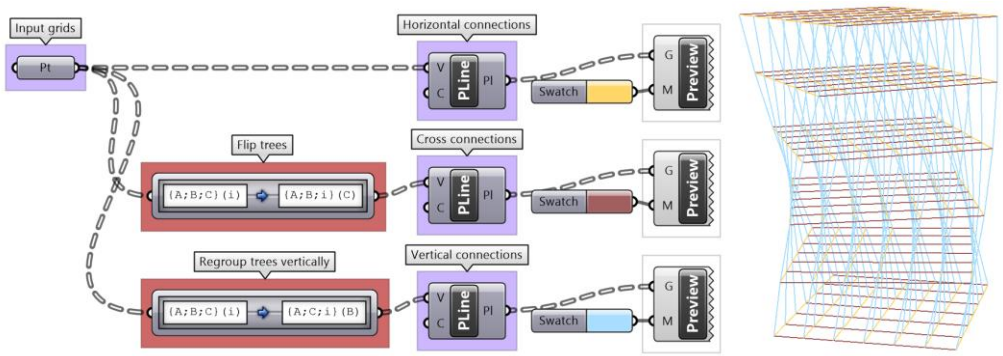


Solución

El árbol inicial tiene 42 ramas, 7 ramas en cada uno de los 6 árboles.



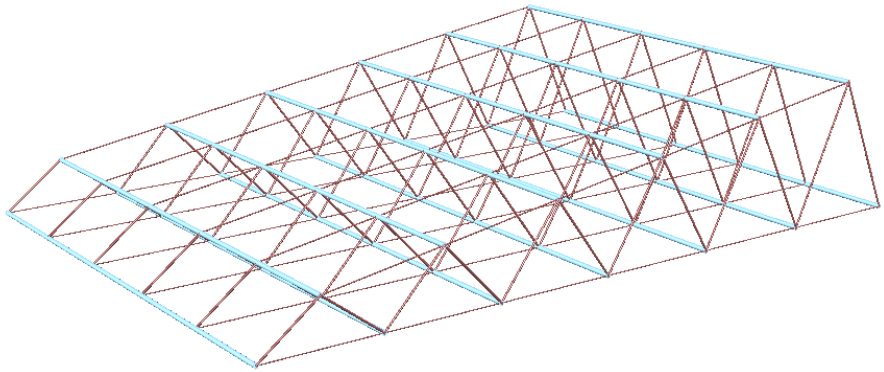
El componente *Polyline* conecta los elementos de cada rama.	
Invierta los árboles con *Path Mapper* cambiando los índices de las ramas y los elementos.	
Reagrupe los elementos de las ramas correspondientes en todos los árboles utilizando *Path Mapper*	

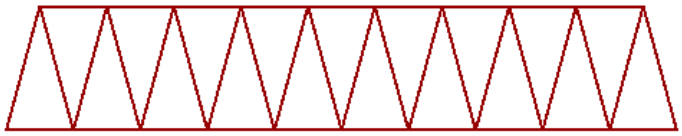
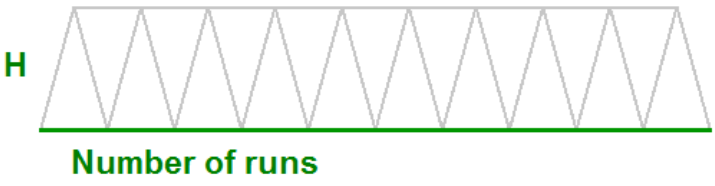
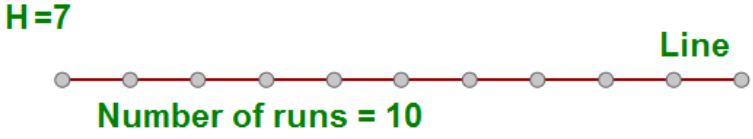
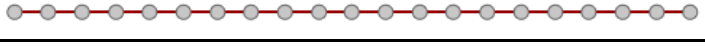
Resultado final: cree todas las conexiones.	

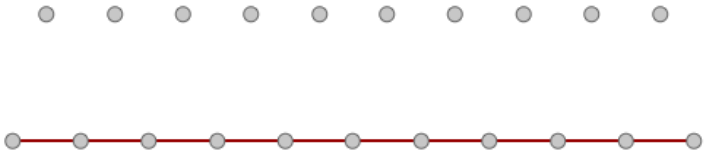
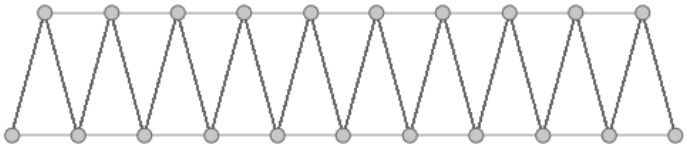
3_7: Tutoriales de estructuras de datos avanzadas

3_7_1: Tutorial de cubierta inclinada

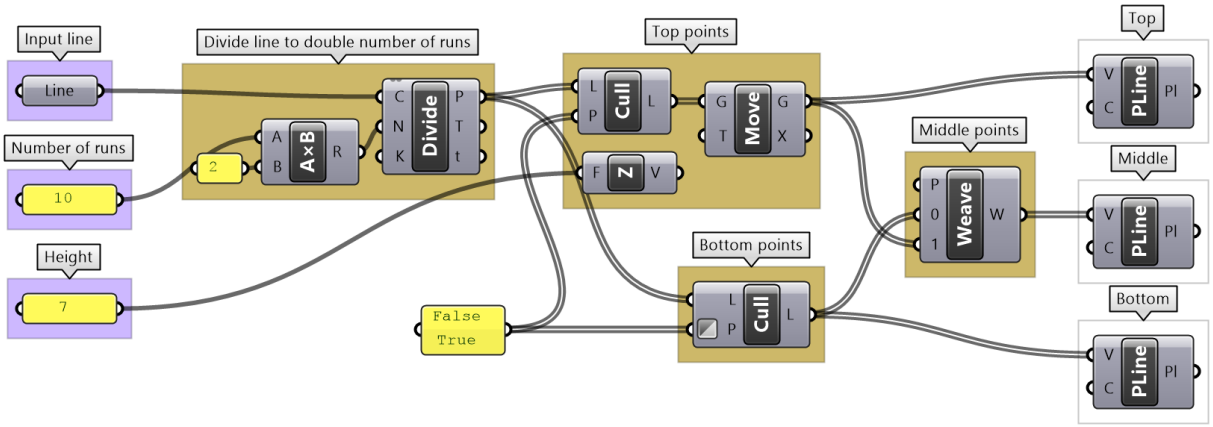
Cree un sistema de celosía paramétrico que cambie gradualmente de altura como se muestra en la imagen.



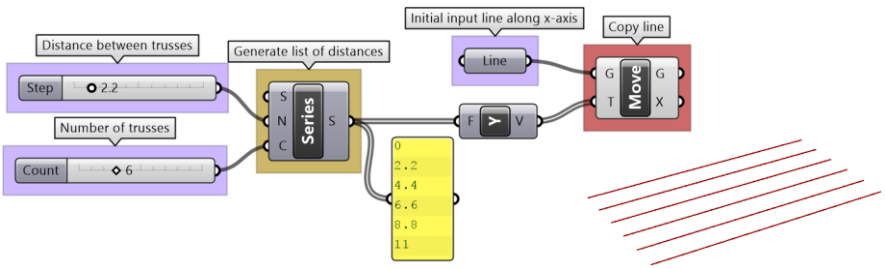
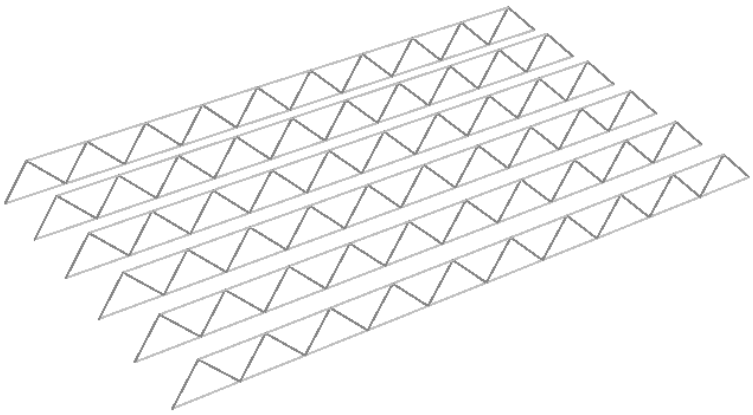
Solución	
Análisis del algoritmo: en primer lugar, cree una celosía simple.	
Identifique la salida que desee para una sola celosía.	
Defina la entrada inicial. 1- Línea base en el plano XY 2- Número de instancias 3- Altura	
Identifique los pasos de los algoritmos:	
Cree la entrada (L=línea, H=altura y R= #instancias)	
Divida la curva por 2*R.	

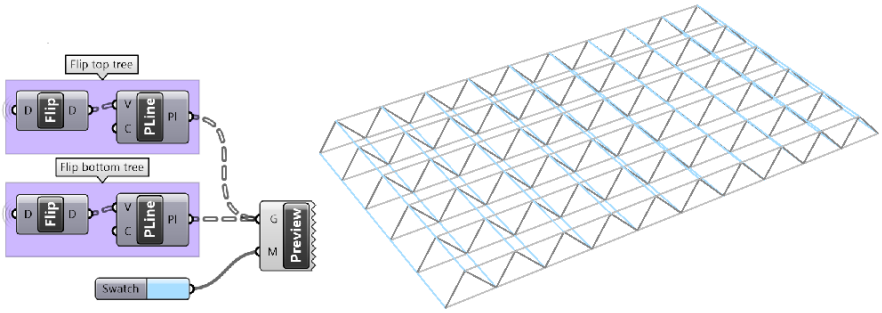
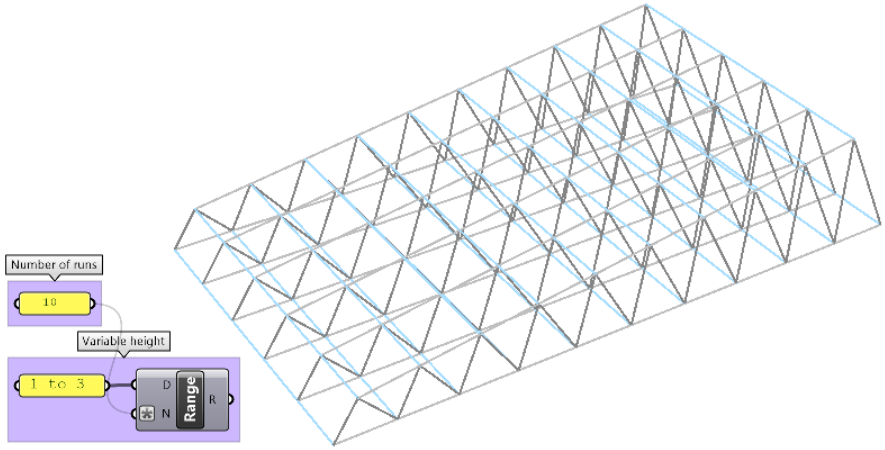
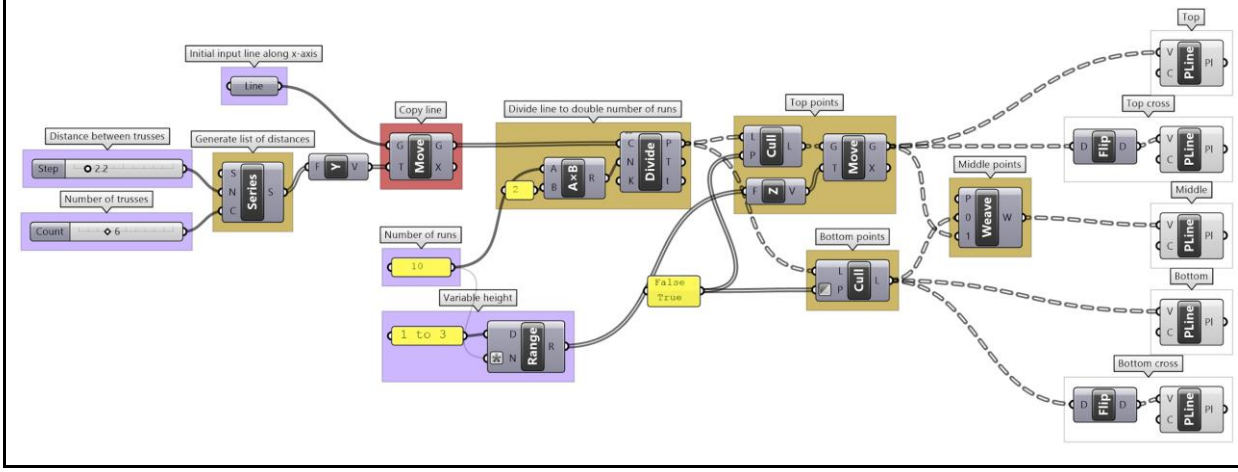
Mueva cualquier otro punto en la dirección Z por altura.	
Cree 3 conjuntos de puntos ordenados para las vigas inferiores, las vigas superiores y las vigas centrales, y luego conecte cada uno de los 3 conjuntos con una polilínea.	

Implemente el algoritmo en Grasshopper



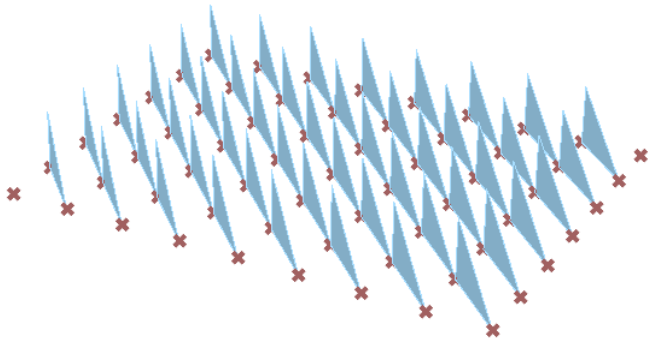
Cree varias celosías con altura variable

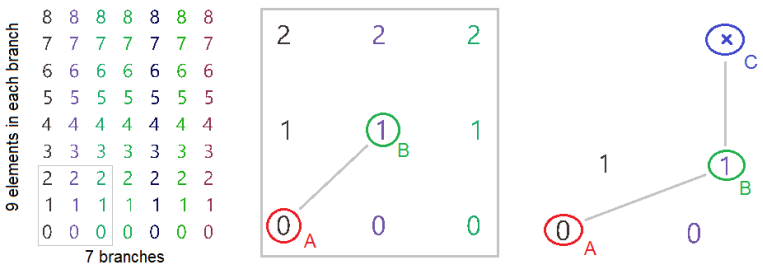
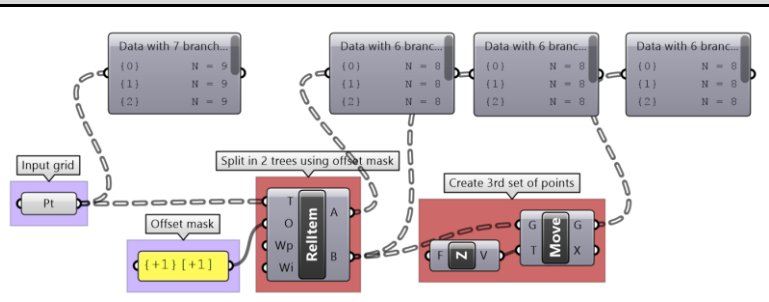
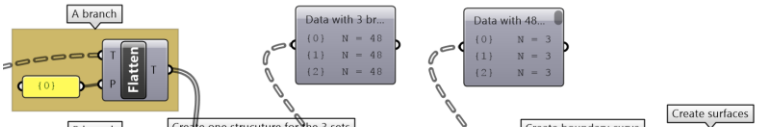
Cree una serie de líneas base utilizando la línea inicial y cópielas en la dirección del eje Y.	
Utilice la lista de líneas como entrada en lugar de una sola línea. Observe que en lugar de una lista de puntos para cada uno de los 3 conjuntos (inferior, superior y medio), ahora tenemos un árbol o rejilla de puntos con un número de ramas igual al número de celosías.	

Cree conexiones cruzadas mediante la operación de árbol ***Flip*** para los árboles inferior y superior.	
Cree una altura variable.	
Implementación de la solución completa en Grasshopper	
	

3_7_2: Tutorial de triángulos en diagonal

Dada la rejilla de entrada, utilice el componente *RelativeItem* para crear triángulos en diagonal.

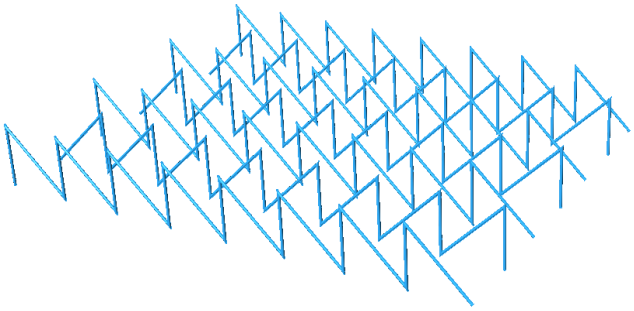


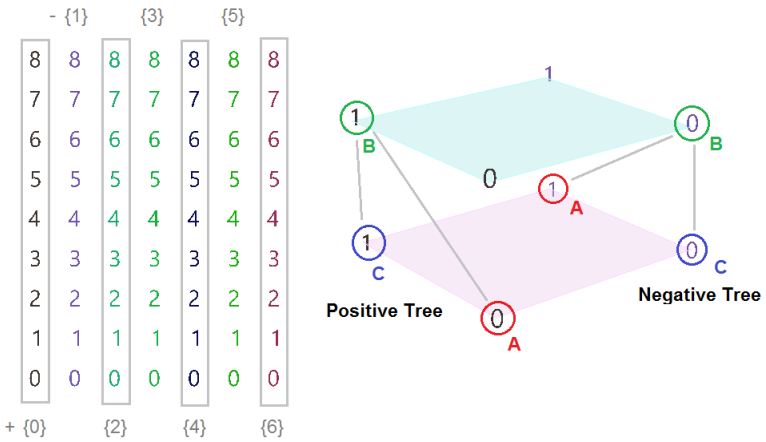
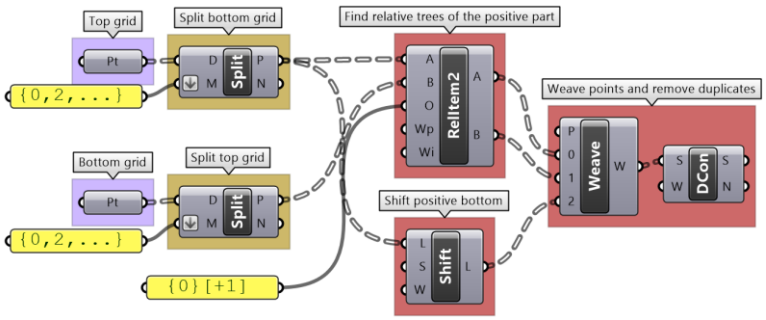
Solución	
Análisis del algoritmo	
Para generar los triángulos, necesitamos 3 conjuntos de puntos de esquina. Dos de los conjuntos de puntos (A, B) están dentro de la rejilla. B está en diagonal respecto a A (el índice relativo es +1 rama y +1 elemento) El tercer conjunto de puntos (C) es una copia del conjunto (B) desplazado verticalmente. Agrupe las esquinas para conectarlas en los límites y luego generar superficies.	
Implementación en Grasshopper	
Utilice el componente ***RelativeItem*** para crear el conjunto A y el conjunto B (utilice la máscara "{+1}[+1]). Mueva el conjunto B verticalmente.	
Cree un árbol con 3 ramas para los conjuntos A, B y C. Invierta el árbol para agrupar los puntos	

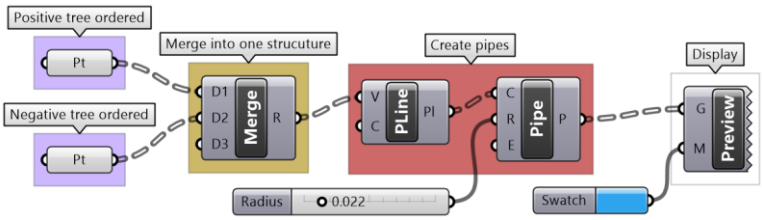
correspondientes. Utilice ***Polyline*** y ***Boundary*** para generar las superficies.	

3_7_3: Tutorial de zigzag

Cree la estructura que se muestra en la imagen utilizando una rejilla base como entrada.

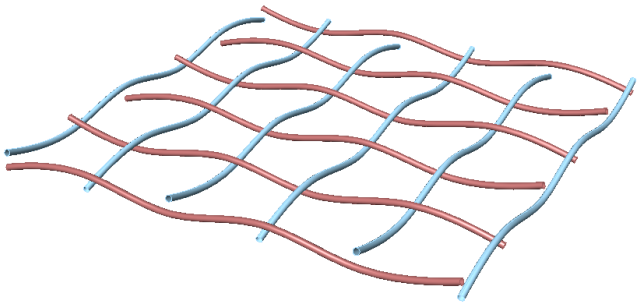


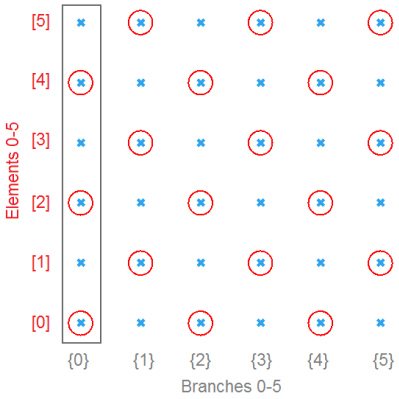
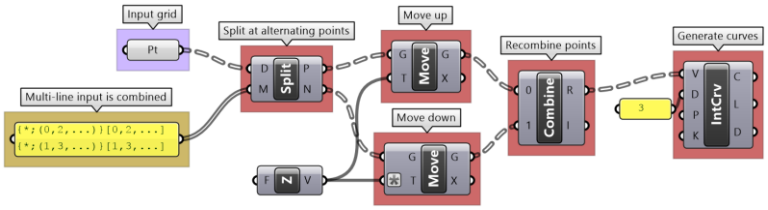
Análisis del algoritmo	
Como los zigzags alternan sus direcciones, es mejor dividir la cuadrícula en 2 partes, positiva y negativa. Encuentre 3 conjuntos de puntos en el árbol positivo y el orden. Invierta los elementos en las ramas del árbol negativo y luego encuentre los 3 conjuntos de puntos y el orden. Vuelve a fusionar los 2 árboles para crear la geometría a través de puntos.	
Implementación en Grasshopper	
Utilice el componente ***Split Tree*** para generar árboles positivos y negativos para las rejillas inferiores y superiores. Utilice la máscara de división {0,2,...}. Utilice ***RelativeItems2*** para crear los árboles A y B, y utilice la máscara relativa {0}[+1]. Utilice ***Shift*** para crear el árbol C. **Entrelace** los datos y elimine los duplicados.	

Fusione los árboles positivos y negativos ordenados para generar la geometría utilizando los componentes ***Polyline*** y ***Pipe***.	

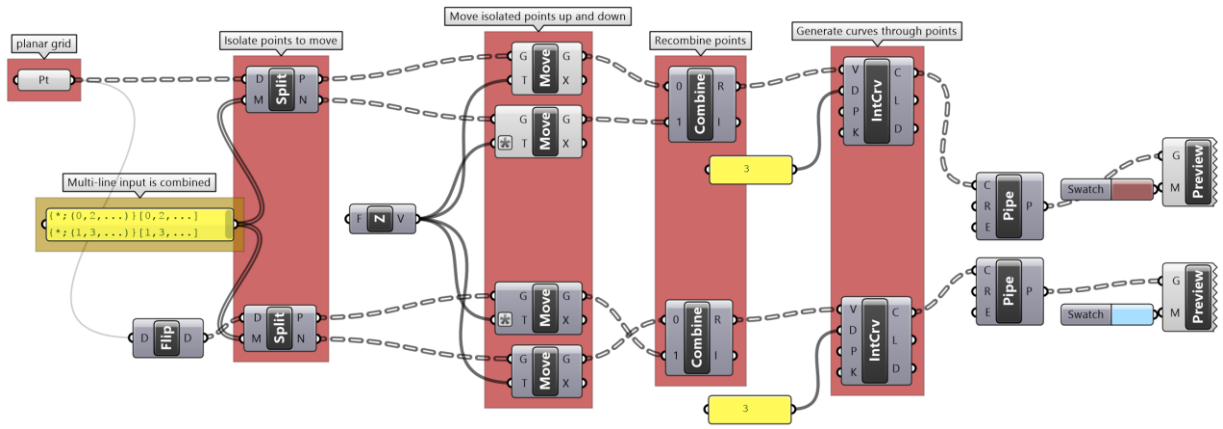
3_7_4: Tutorial de tejido

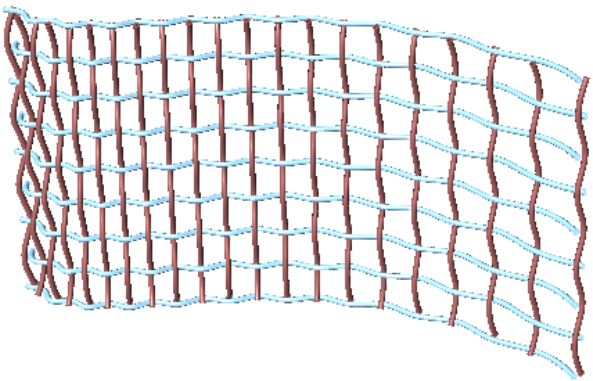
Cree hilos de tejido plano utilizando una rejilla rectangular como entrada inicial. Defina la densidad y el tamaño que desee. Paso adicional: Haga que el tejido se extienda a lo largo de cualquier superficie.

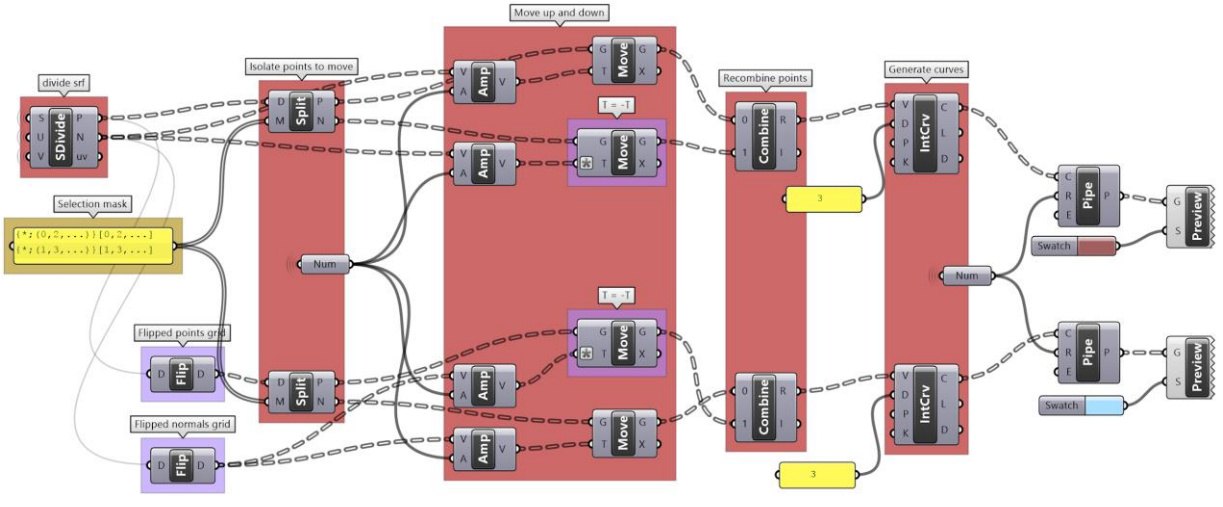


Análisis del algoritmo	
La entrada es una rejilla cuadrada plana con ramas verticales. Para los hilos verticales: Divida la rejilla en dos partes alternando elementos en cada rama. Mueva la primera parte hacia arriba y la segunda hacia abajo; luego recombine las partes en un conjunto. Dibuje una curva a través de los puntos en cada rama. Invierta la rejilla y repita los pasos para crear curvas horizontales.	
Implementación en Grasshopper	
Utilice ***Split Tree*** para separar los puntos alternativos y moverlos hacia arriba y hacia abajo. ***Combine*** para combinar los puntos y ***IntCrv*** para interpolar a través de los puntos de cada rama ***Flip*** para invertir el árbol y repita ***Split*** , ***Combine*** e ***IntCrv*** para crear curvas en la otra dirección	

Solución completa en Grasshopper





Solución adicional

En lugar de utilizar el eje Z para mover los puntos hacia arriba y hacia abajo, utilice la dirección normal de la superficie en cada punto.
Nota: Asegúrese de que la estructura de datos de las normales y de los puntos coinciden